

Jim Draft

Welcome to Macintosh Technical Support!

Your ID:

SUPT.

Your KEY:

Welcome to Macintosh Technical Support. We provide developer support through Apple's electronic mail system on TYMNET. Using this system increases our efficiency, which translates into better support for developers.

The attached document is the first draft of the user's guide for the electronic mail system. The last page is a list of the TYMNET phone numbers, sorted by state. To use the system, you'll need a computer (we suggest an Apple), a modem, and a phone. You pay for the telephone call, and Apple pays for the electronic mail system charges, with the understanding that the system is to be used for technical support matters.

The first time you log onto the system, you should change your key (password). Change it to something that you'll remember, and something non-obvious. Please keep it secret. If you have problems, or suspect unauthorized use of the service, send me a message, or give me a call. For information on changing your key, type

: READ ** CHANGE . KEY

When you have a question or problem, send a message to me at the MAC mail station. The mailbox is usually checked 3 or 4 times a day (I check it from home on the weekends), and we will try for 24 hour turnaround at worst.

I'd appreciate comments on the document (this is a draft), and on the system. Let me know what kind of equipment you're using to access the system, and how it works for you.

OLD USERS OF THE SYSTEM

If you've been using the system for a while, you'll notice a couple of changes. The account name is now SUPT (it used to be APPLE). You send mail to MAC rather than MACTECH. Phone numbers are the same.

1200 BAUD USERS

If you're experiencing trouble at 1200 baud, and your software uses XON/XOFF protocols, be sure and send CONTROL-X CONTROL-R before EMSAPP in talking to the system. This lets their software know you're using XON/XOFF for flow control.

Bob Martin

Mac Tech Support

racing 2

Company of the compan

one con a lega comen ato de colle a como ago atomo ago al comen a como a

u no como los comes en estra hiporia por la sunta como tro sul proceso. El como estra como como en estra de la Entra de la secución de la como de la como en estra co En resolución por entra como en estra como en

in the light of the company of Association and the lifety to a substantial and the company of th

ELECT MAIL

Apple Computer, Inc. 20525 Mariani Avenue Cupertino, CA 95014

Macintosh Tech Support Electronic Mail System Users Guide

Table of Contents

Section	Page
Introduction	1
Preparing and Sending messages	2
Mail management The "IN" list The "OUT" list How to receive Message retrieval Cancelling a message Control characters	6
Accessing the Electronic Mail System	7
Leaving the EMS	8
Error messages and trouble reporting	9
ONTYME II command reference	11
Using Access /// on the EMS	12
Using Micro-Courier on the EMS	13

* CARA TRANSITION OF ACTIONS TO THE ACTION OF THE ACTION O

and the second of the second o

weigned in the second of the s

The posts of the company of the state of the

of the following state of the same of the

Introduction

This document is a users guide for Apple computer third party developers using Apple's electronic mail system (EMS). Apple Computer, Inc. uses TYMNET's "ONTYME II" message switching network with Apple][and Apple /// computers to create an effective and efficient computer based message network.

The ONTYME II message switching system operates on a store and forward basis. It accepts messages for transmission, and then either stores them while waiting for users to make contact with the service, or delivers the messages to on-line designated hardcopy printers. The network is supported by Technical Support.

Messages (mail) are created off line using either an Apple][or Apple ///, and then entered into the electronic mail system.

This service is provided to third party developers to improve communication and productivity between Apple and outside developers.

Each station or user in the Apple account is identified with a unique call directing code (CDC) and password.

All commands entered into the system are preceded with a colon (:), to indicate that they are commands and not text. Commands must begin at the left margin (column \emptyset).

The following definitions and conventions are used in the examples given in this guide. Refer to page 11 for the ONTYME II command definitions.

Convention used	Definition				
CDC, CDC1, CDC2, etc	A call directing code (a user)				
Underlined text	Commands typed by the user				
Regular text	Text or messages printed by EMS				
(cr)	A carriage return typed by the user				
(s)	A space that is required				
EMS	Apple's electronic mail system				

Questions or suggestions regarding EMS should be forwarded to EMS station "MAC".

Preparing and Sending Messages

The :SEND command takes the text in your workspace and sends it to the CDC (or CDCs) that you specify.

Managing the Workspace - : TYPE and : ERASE commands

Anything that is not recognized as a command goes into your workspace as part of the message you want to send. So, if you type "; IN" rather than ": IN", the erroneous command goes into your workspace. If you are unsure of what text is in your workspace, use the :TYPE command to display it. The :ERASE command clears the workspace.

Standard Message Format

All messages should be created and edited off line, using a text editor of your choice. This saves valuable connect time on the EMS.

When preparing a message, do not exceed a 68-character line length. By keeping your lines under 68 characters, you insure that international locations will be able to read each line in its entirety. Although there have been recent improvements, most international locations still use the older telex type equipment which has a 68-character line limitation.

Text editors such as Applewriter and the Pascal editor let you set up a "boilerplate" file that includes a right margin (or line length) selection of 68 characters. Using a boilerplate file is a good way of insuring a consistant format.

All messages should be prepared using the following format:

Format	Example
Date	3/1/83
Addressee	To: MAC tech support
Sender	Fr: Bob Martin "MAC1023"
Text	This is the new standard message format for a message sent through the ONTYME or telex networks.
End of message and operators initials	END/RTM
Send command	:SEND(s)MAC(cr)

The :SEND command packages up the text in your workspace and sends it to the destination CDC. If you want to send the message to more than one CDC, separate the CDCs by at least one space.

General information

The Apple electronic mail system provides tools that allow you to manage your mail box effectively and efficiently.

and the second of the second

ATTENDED TO CHEST

工物 位于 医类性 克里克

19:322 : 195

These tools are:

- I. The "IN" list
- II. The "OUT" list
- III. Receiving commands
- IV. Message retrieval
- V. Cancelling a message
- VI. Control characters

I. The IN list

Each time someone sends a message to you an entry is made into a list for all messages waiting to be received by your CDC. This list is called your "IN list". When you actually ask to "READ" a message, the entry in the "IN list" is removed, and entered into the "IN OLD list".

To see if any messages are in your mail box waiting for you; you examine the "IN list" by issuing this command:

: IN (cr)

If no messages are waiting, then EMS will respond with "NONE". Otherwise, the following information is transmitted to you:

SENDER	SENT	MSG#	LENGTH
aaa	ddmmyy hh:mm	bbbb	ccc

aaa = Sending CDC

ddmmyy = Day, month, year that message was sent

hh:mm = Time message was sent (your local time)

bbbb = Message number assigned by the EMS

ccc = Number of characters in the message

I. The IN list

To examine the list of messages you have received and read in the last five days, issue the command:

:IN(s)OLD(cr)

The following information is transmitted to you:

SENDER SENT MSG# READ

aaa ddmmyy hh:mm bbbb dddd

The legend is similar to the in list, except (dddd) indicates the date and time the message was read, and omits the message length.

Now, if you want a message to be re-transmitted to you, type:

:READ(s)bbbb(cr)

II. The OUT list

Each time you send a message, it is entered on a list of all messages you've sent, called your "OUT list". When the recipient of a single addressed message has read it, the entry is removed from your "OUT list" and placed in your "OUT OLD list" where you have confirmation of its receipt, plus the date and time it was read. For messages sent to multiple CDC's an asterisk (*) will be added before each location which has received it. When all locations have received the message, the entry will be moved to the "OLD OUT list".

To see if there are messages waiting to be delivered, examine the "OUT list" by issuing the command:

:OUT(cr)

If all your messages have been received, the EMS responds with "NONE". Otherwise, the following information is transmitted:

NAME SENT MSG#

aaaa ddmmyy hh:mm bbbb

aaaa = Receiving CDC

ddmmyy = Day, month and year message was sent
hh:mm = Time message was sent (your local time)
bbbb = Message number assigned by the EMS

Although EMS keeps lists of your unread outgoing messages for up to 14 days after transmission, it's a good habit to check your "OUT list" at least once a day.

III. Receiving commands

If you are a dial-up user you must check your mail box for incoming mail (as you do at home). It's recommended that this be done daily.

The second

PARENTE CO

To receive your mail from the ONTYME II system, you must use one of the "READ commands" listed below:

But, before entering read commands, make sure you are ready to record the mail you will be reading either in memory, or on disk.

COMMAND	Purpose
:READ(cr)	Transmits the oldest message on your IN list, then removes it from your IN list and removes your CDC from the
	senders OUT list.
:READ(s)ALL(cr)	Transmits all the messages on your IN list, removes them from your IN list and removes your CDC from the senders OUT list.
:READ(s)message number(cr)	Transmits a specific message re- cently received to your CDC. This only works if the message is still on either your IN list or OUT list.

IV. Message retrieval

Any message still on your IN or OUT lists can be retrieved by issuing the following command:

:GET(s)message number(cr)

This command does not automatically transmit the retrieved text; the following command should be used to transmit the retrieved text:

:TYPE(cr)

V. Cancelling a message

You can cancel any message sent via your CDC that has not been read by all recipients (if some recipients have already read the message, ONTYME will notify you). "Message number", below, is the master message number that ONTYME assigned to the message at transmission.

:CANCEL(s)message number(cr)

VI. ONTYME II Control Characters

EMS uses these control characters:

- (CTRL) H Deletes the last character typed.
- (CTRL) S Stops the data being received.
- (CTRL) Q Starts the data being received.

CTRL-H is the left arrow on most keyboards. Typing CTRL-S suspends output, but it may take 20 characters or so before the EMS recognizes it. To start things going again after CTRL-S, type CTRL-Q.

Accessing the Electronic Mail System

Accessing EMS is done by placing a telephone call to the local TYMNET office (see attachment to determine local number) and then "logging" into the EMS computer using the "log in" instructions listed below.

TYMNET/ONTYME II log in instructions

Proper connection with TYMNET has been made when the following appears on your screen:

Respond by typing: \underline{A}

This lets TYMNET know what terminal speed you're using. The system will respond with:

"please log in:"

Respond by typing: emsapp(cr)

The system will respond with:

">ONTYME II date time GMT"

"ID?"

Respond with:

 $\underline{Apple.xxx(cr)} \qquad (xxx = users CDC)$

The system will respond with:

"KEY?"

Respond by typing your key (password), (it shouldn't echo) followed by a carriage return (cr). The system will respond with:

"ACCEPTED" (please refer to page 9 if "ACCEPTED" sign does not appear).

A sample log in sequence is shown below: xxxxxxx000000xxxxxx00000(garbage)

A -1326-023please log in :EMSAPP(cr)
id? SUPT.MAC(cr)
KEY? non printing password is typed (cr)
ACCEPTED

Leaving EMS

You normally end your EMS session by issuing the command ":QUIT". If there are any messages still waiting in your IN list, the following message will be printed:

"MESSAGES WAITING:" - if your IN in list is not empty

Pressing RETURN (cr) in response to the above message ends the session. Any other response will ignore the quit command, leaving you connected to EMS.

For example:

:QUIT(cr)
MESSAGES WAITING: (cr)
DROPPED BY ONTYME II
01 MAR 83 11:47:35

please log in: (message will appear when you have successfully terminated from ONTYME II).

Note: When using Access /// you must press "OPEN-APPLE Q" to exit, after you have left ONTYME II.

To leave the system immediately, use the command:

:LOGOUT(cr)

Error Messages

ARLAG

Symptom or message	Action required
Ring no answer	Call message network supervisor.
Busy signal when call placed	Wait 5 minutes and try again.
"ERROR TYPE USER NAME"	Re-type user name.
Terminal prints DDOOUUBBLLEE	Insure that the Apple/software is set for full duplex.
"HOST DOWN"	Wait 15 minutes and try again.
"please log in" printed during session	Communications failure: log in again and re-start job from point of last accepted message.
"all ports busy"	Wait 5 minutes and try again.
"all circuits busy"	Wait 5 minutes and try again.
"no such recipient"	"name" printed station name (CDC) is not a valid station.
"invalid command"	Command misspelled.
"message not on in list"	Message requested is not on IN list or the IN OLD list.
"all messages read"	The IN list is empty.
"group code file not found"	Group code was either mistyped or

NOTE: Error messages with quotes will appear on screen when you are logged onto EMS.

"invalid message number"

"invalid user"

non-existent.

entered.

Message number with an error was

is not valid in the system.

Prints when the specified "username"

Error Messages

Symptom or message

Action required

"message #??"

Prints when an invalid message number is entered in GET or READ commands.

"message not on out list"

Prints if user tries to cancel a message that has already been read.

In general, if you are having trouble getting logged into ONTYME II, or having trouble with the command formats, you should contact Mac tech support at (408) 973-2282.

Error Messages

ONTYME COMMAND REFERENCE

Section 1

ugust 23,	1982	t bid
Telephone :	number of Ontyme:	L. R. C. Sales C.
All command	is are preceded by : (colon)	
:IN	This checks the mailbox for any messages. Messages are listed in order of the oldest message first.	
:IN OLD	This lists the messages for the last 14 days that have already been read.	
:OUT	This lists any messages sent that have not been read by the recipient.	
:OUT OLD	This lists the messages for the last 14 days that have already been read by the recipient.	
:READ	This reads the message in the mailbox. If there is more than one message, than the oldest is read first.	
:READ ALL	This reads all messages in the mailbox.	
:CANCEL	Use this command to cancel a message. Type :CANCEL (#).	
:SEND	This command followed by the mailbox sends a message.	
:SEND CC	This command followed by mailbox(es) sends a message with carbon lists attached.	
:SEND RUSH	This command followed by mailbox(es) sends a message immediately to recipient(s) who have available dial-out stations.	
:GET	Retrieves a message recently sent or received via your ID, if the message is still on one of your in or out lists.	
:TYpe	Type displays the text in your workspace.	
:ERASE	Deletes all text in the workspace.	
:KEY	Initiates the change-key (password) prompt series.	
NOTE: This (READ(s)**(display is available on the EMS by issuing the command s)COMMANDS	

:READ ** HELP

- Before you log into the electronic mail system, you should have your message(s) created and stored on disk ready for sending. At this time boot Access ///. When you see the first menu, select terminal mode, then press return, and you will see only a cursor on the screen. Now place a call to the local TYMNET office and follow the instructions on page 7.
- 2) You must set up a recording file for your message to be stored.

 to do this press OPEN-APPLE S. Select "Change the recording file"

 with the up/down arrows and press return. Access // will ask for the
 file name, or the new file name if you are changing the recording file.

 If you are operating at 300 or 1200 baud and want to use a Silentype as
 the file it would be ".SILENTYPE". If you want to record to disk, name
 the file with the path first then the name (i.e. .D2/EMSLOG) and the
 message will be recorded to disk. When saving messages to disk, you
 may want to change the recording filename to avoid writing over a
 previous message.
- 3) To record to disk or (Silentype), you must turn on the recording file by typing OPEN-APPLE R. The cursor starts flashing. The message will be sent to the recording file. After the message is recorded you must turn off the recording file so that the message buffer in the Apple is cleared. To close the recording file press OPEN-APPLE R again, just as you did to turn it on. You now see that the cursor is not flashing. Follow the "mail management" instructions on checking your mail box for messages and reading messages.
- 4) To send message(s), return to the Access /// set up mode by pressing OPEN-APPLE S. Use the up/down arrow to select "Exit terminal mode" and press RETURN. Select "Transmit a file" and press return. Enter the pathname (i.e. ".D2/Apple") for the file to be sent. Access /// now asks for delay parameters. The following normally work satisfactorily:

line delay Ø(cr)

character delay 8(cr)

- 5) After transmitting the message, Access /// responds "File transmission complete". To return to terminal mode press up arrow twice and then RETURN. If you put the :SEND command at the end of the message you should see a message number. It's a good practice to write these down, in case you need to refer to a specific message again. Enter the send command now, followed by return if you did not embed it into your text.
- 6) After sending and receiving all mail, log off ONTYME II (refer to page 8).

Using Micro-Courier on the EMS

- 1) Load Micro-Courier
- 2) When the main menu appears, type "1" and press RETURN.
- 3) When the editing menu appears, type "1" or "2" and press RETURN.
- 4) Prepare or edit your messages using the correct format (see page 2).
- NOTE: More than one message can be prepared per file using Micro-Courier. The important thing to remember is not to forget the send command after preparing each message.
- 5) After message(s) have been prepared and edited, press the ESC key.
- 6) Micro-Courier will ask for file name.
 - a) Type month day/message # (use the first msg#)

example: MARØ1/ØØ1

- b) After typing the file name, press RETURN.
- 7) Micro-Courier will ask which drive to store file.
 - a) Type either "1" or "2" (in most cases it is "2")
 - b) Press RETURN.
- 8) Press the ESC key to leave storage area.
- 9) Press the ESC key again to go back to main menu.
- 10) Type "6" and press RETURN.
- 11) Type "l" and press RETURN. Micro-Courier will ask for phone number. Type the local TYMNET number.
- 12) Type "3" and press RETURN. Type "6" and press return key.
- 13) Type "7" and press RETURN.
- 14) Type "5" and press RETURN. Micro-Courier will ask for out file name. Type name created in step 6. After typing file name, press RETURN. Micro-Courier will ask which drive; type the same number as in step 7, then press RETURN.
- 15) Type "6" then press RETURN. Micro-Courier will ask which file name will store incoming traffic. Type month day/in msg#.

example marØl/inØØl

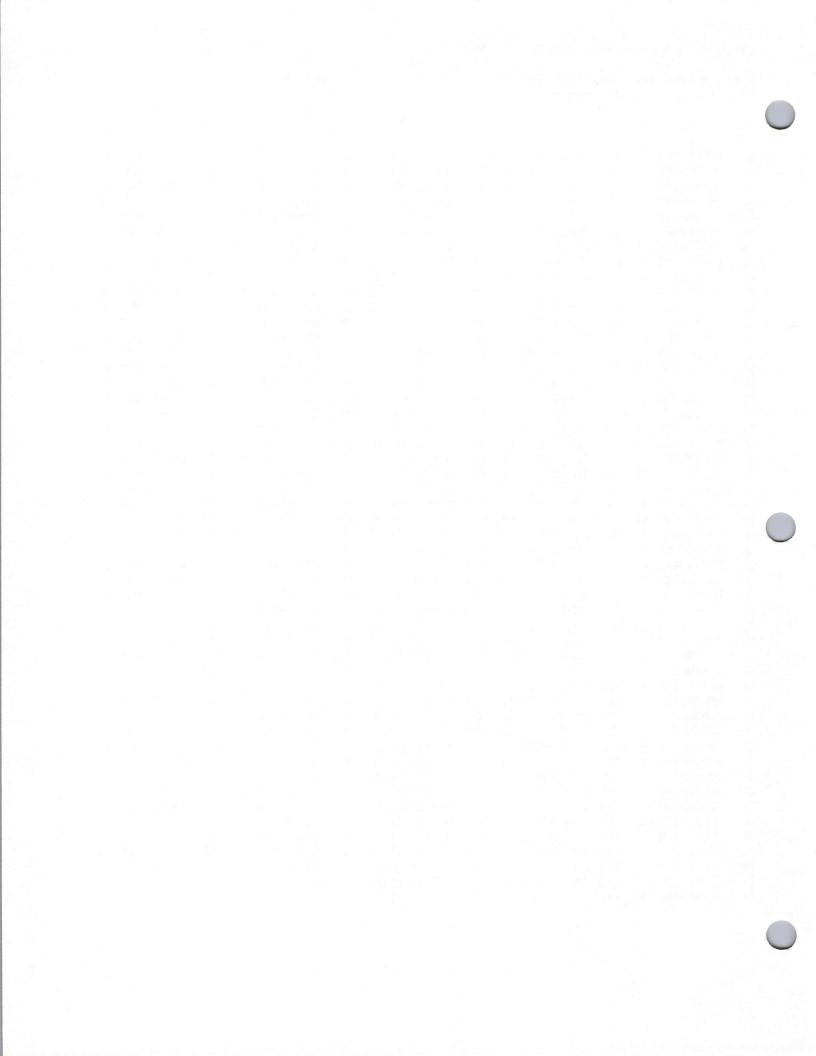
Press RETURN. Micro-Courier will ask which drive will store messages. Indicate either 1 or 2 and press RETURN.

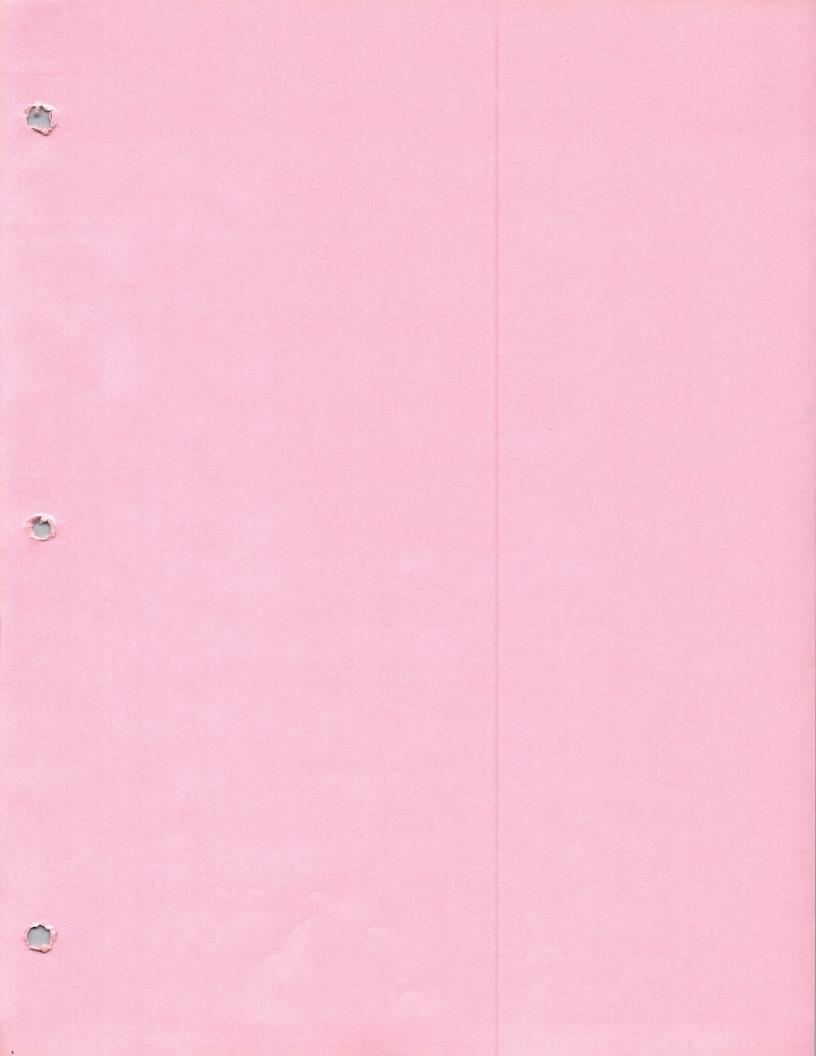
Using Micro-Courier on the EMS

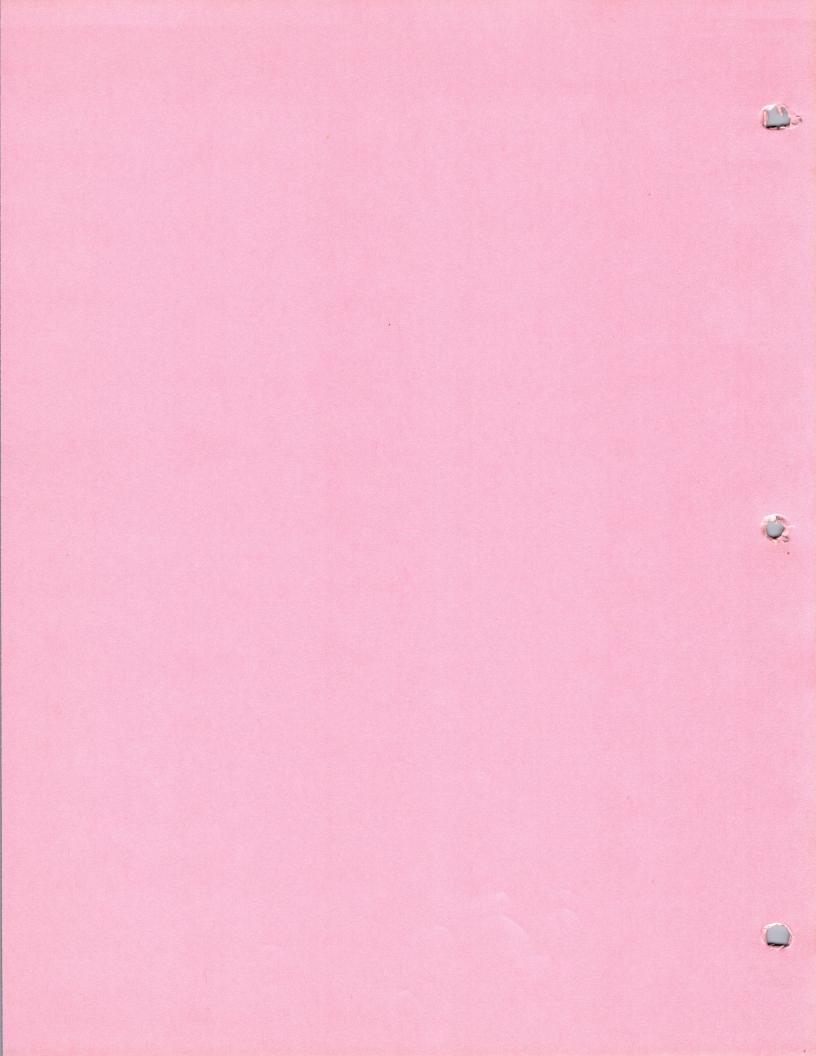
- 16) Connecting to EMS
 - a) Type "1" and press RETURN.
 - b) Log in to TYMNET/ONTYME (see page 7).
 - c) When you have logged onto ONTYME/and have received the "accepted" acknowledgment, type ":LOAD ON" and press RETURN.
- 17) How to send
 - a) When you have received the accepted acknowledgment press the ESC key and "T" key. Micro-Courier is now sending the out file to ONTYME.
 - b) As messages are accepted, ONTYME will send message numbers back to you which will show on screen.
- 18) How to receive
 - a) Once sending has been completed, you can receive incoming mail. Press the ESC key and "R" key. This tells Micro-Courier to store all messages received in the file that was named in step 15.
 - b) Type ": READ ALL" and press RETURN. ONTYME is now sending your incoming mail.
- 19) After sending and receiving mail, log off ONTIME (refer to page 8).
- 20) Press ESC and "E". Type "4" and press RETURN.
- 21) Type "8" and press RETURN. Micro-Courier will go back to main menu.
- 22) How to print incoming messages
 - a) Type "1" and press RETURN.
 - b) Type "2" and press RETURN.
 - c) Type same file as was indicated in step 15. Type drive number.
 - d) Press ESC key.
 - e) Type "3" and press RETURN.
 - f) When printing has been completed press ESC. Micro-Courier will go back to main menu.
- 23) Type "8" and press RETURN.

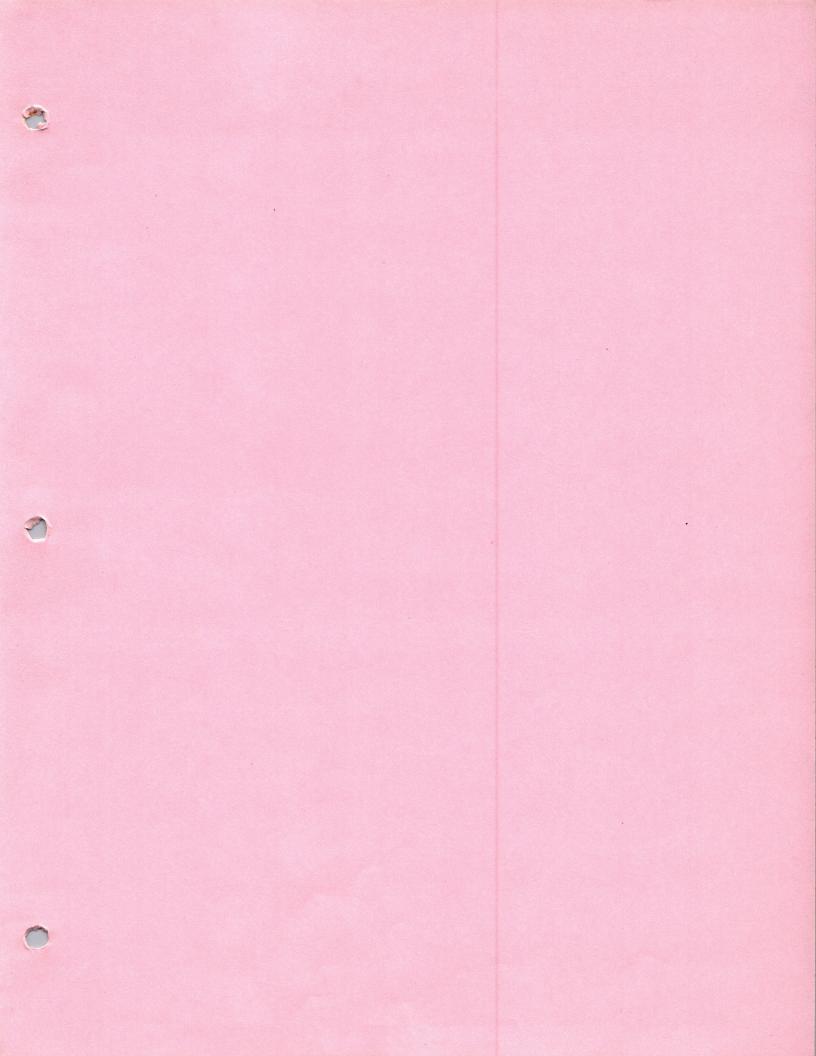
H, M, L, I = High, Medium, Low, International capacity 3=380 band only, 300/1290 band otherwise

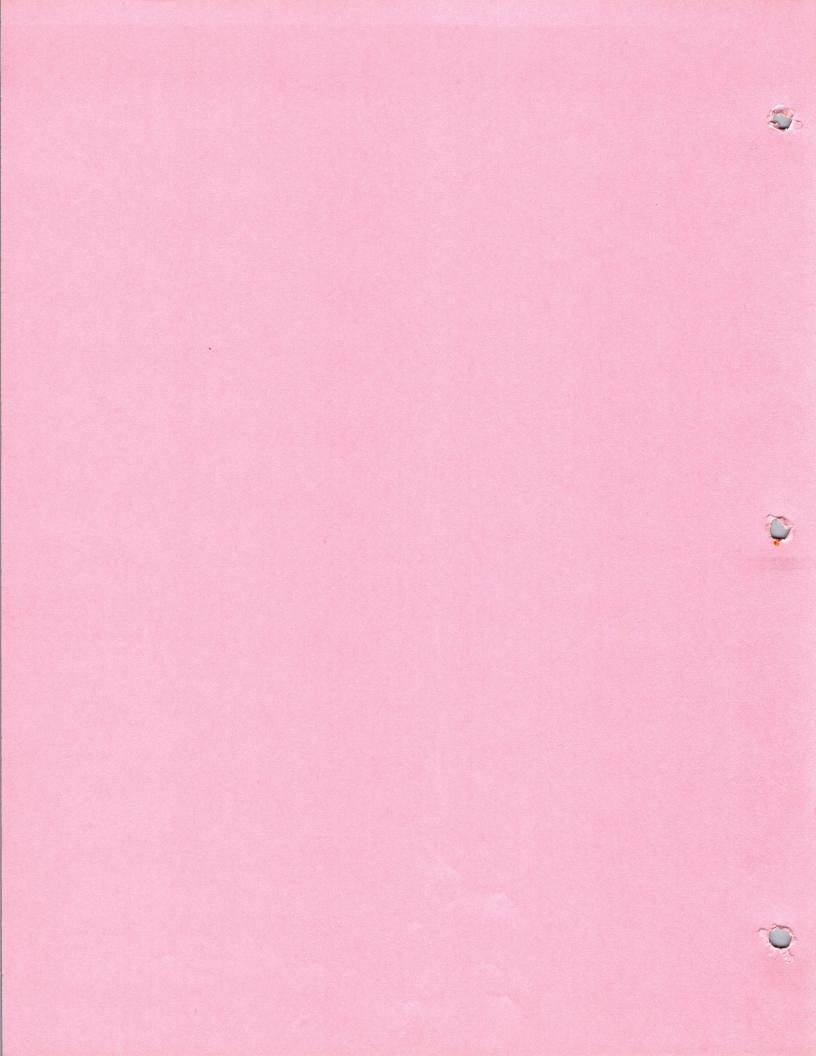
	State	Phone	Type												
	AL	205-432-338		CA	805-682-9641	H	KAN	913-233-1682	L	NH	603-623-8855	L	RI	401-273-0200	H
	AL	205-834-341	0 L	CA	916 -448-8 151	H	KAN	913-384-1544	H	NH	603-882-0435	M	SC	803-252-0840	M3
	AL	205-882-300	3 M	CO	303-475-2121	M	KY	502-499-7110	H	NH	603-893-6200	L	SC	803-271-2418	M3
	AL	205-942-414	-	CO	303-830-9210	H	KY	686-253-3463	H3	NJ	201-432-0792	L3	SC	803-271-9967	M3
		907-278-351		CT	203-227-7189	-	LA	318-237-9580	H	NJ	291-460-9180	H3	SC	883-577-2179	L
		907-586-661		CT	203-242-7140	H3	LA	318-688-4666	L	NJ	201-483-5937	H3	SC	803-585-2637	L
	ARK	501-372-578		CT	203-367-6021	H	LA	504-291-2650	H	NJ	291-785-4480	H3	TN	615-367-9382	H
	ARK	501-782-321		CT	203-755-1153	L3	LA	504-524-4371	H	N	201-894-8250	M	TH	615-637-3118	H
	AZ	602-254-581		CT	293-789-9579	M3	MA	413-781-6830	H	NJ	201-981-1900	H	TN	615-756-5856	M3
	AZ	602-790-076		CT	203-965-0000	H	MA	616-482-5605	H3	NJ	689-235-3761	M3	TN	981-529-0170	H3
	CA	209-268-121		DC	783-442-3908	H3	MA	617-482-4677	H3	NJ	609-452-1018	H	TX	214-263-4581	H
	CA	209-577-560		DC	703-442-3968	H	MA	617-482-7035	H	NJ	609-452-1018	H	TX	214-638-8888	H
	CA	213-203-986	_	DC	703-691-8200	H	MA	617-755-0916	L	NH	505-843-6301	H	TX	214-758-1756	L
	CA	213-280-110	-	DC	703-734-3900	H3	HD	301-547-6100	H	M	212-269-9642	H	TX	512-225-6002	H
	CA	213-331-354		DEL	302-429-0112	L	MD	301-770-1680	M3	MY	212-532-0437	H3	TX	512-444-3280	H
	CA	213-365-201		DEL	302-678-0449	L	ME	207-774-2654	H	MY	212-685-4414	H3	TX	512-883-8050	H
	CA	213-435-708		FLA	305-467-3807	H	MI	517-487-2040	H	NY	212-785-5400	H	TX	713-427-5856	L3
	CA	213-518-377		FLA	305-624-7908	H	MI	616-385-3150	H	MY	315-437-7111		TX	713-632-2589	L3
	CA	213-572-099		FLA	305-627-5410	H	MI	616-429-2568	L	NY	516-549-2780	M	TX	713-975-0500	H
	CA	213-574-763		FLA	305-725-8011	L3	MI	616-459-5069		MY	516-872-6500	M	TX	713-977-4080	НЗ
	CA	213-574-683		FLA	305-851-3530	НЗ	MI	616-723-8373	L	MY	518-463-3111		TX	806-762-0136	L3
	CA	213-577-869		FLA	813-535-6441	M3	MI	616-775-1261	L	NY	607-257-6601	M	TX	915-533-1453	H
	CA	213-865-055		FLA	904-252-4481	L3	MI	616-946-0002	L	M	607-962-4481	M	TX	915-563-3745	L
-	CA	213-906-845		FLA	984-434-0134	L	HICH	313-459-8900	МЗ	NY	716-248-8008	H	TX	915-683-5645	M
-	CA	213-986-950		FLA	904-721-8100	M3	HICH	313-569-8350	НЗ	NY	716-295-6691	L3	UTAH	881-364-0788	H
	CA	213-998-046		GA	912-236-1904	L	MICH	313-665-2626	Н3	NY	716-845-6610	H3	VA	703-345-4730	L
	CA	213-998-333		6A	912-352-7259	3	MICH	313-963-3388	Н3	NY	914-328-9580	H	VA	703-691-8200	H
	CA	408-426-840		ID	208-343-0404	H	MICH	517-787-9461	M3	NY	914-471-6100	L	VA	804-528-1903	L
	CA	408-443-433		ILL	217-753-7905	H	MIM	612-339-5200	НЗ	M	914-684-6875	H	VA	804-596-7609	H
	CA	408-980-810		ILL	309-673-2156	L3	MISS	601-769-6502	M3	OH	216-535-1861		VA	804-744-4869	H
	CA	415-462-890	-	ILL	312-346-4961	H3	MISS	601-944-0860	M	OH	216-744-5326	L	VA	804-744-4860	H
	CA	415-490-736		ILL	312-368-4607	H3	MO	314-421-5110	НЗ	OH		H	VA	804-872-9592	
	CA	415-778-342		ILL	312-368-4700	H3	MO	314-731-2304	L3	OH	513-489-2180	H	VT	802-658-2123	r.
	CA	415-785-343		ILL	312-438-5003	F3	HO	314-875-1290		OHIO	419-243-3144	M3	WA	206-285-0109	
	CA	415-798-209		ITT	312-790-4400	H	MO	417-782-3037	Ļ	0H10	614-421-7270	H3	WA	206-473-7810	Ļ
	CA	415-834-870		ILL	815-233-5585	L3	MO	417-931-5044	L	OK	405-947-6387	H	WA	204-754-3900	Ļ
	CA	415-932-011		ILL	815-398-6090	M3	HO	816-232-1897	_	OK	918-582-4433	H	WA	206-825-6576	L
	CA	415-966-855		IND			HO	913-384-1544		OR	503-226-0627		WA	509-375-3367	
	CA	415-986-820		IND	219-424-5162 219-769-7254		HONT	406-494-4637		OR	503-399-1453		WA III	509-747-4105	
	CA	619-296-337		IND			NC	784-376-2545		PA	215-269-9861		WI	414-235-1082	
	CA CA	619-329-077 619-485-199		IND			NC	919-323-4202		PA DA	215-337-9900		WI	414-437-9897	
		619-727-601		IND	317-662-0091		NC	919-379-0470		PA PA	215-432-1500 215-666-9190		WI	414-632-3006	
	CA			IND	812-425-5211	L	HC	919-549-8952		PA			WI	414-722-5580	
	CA CA	707-575-016 714-370-120		IOWA	319-233-9227		NC	919-725-9252		PA	412-765-1320	H	UI UI	414-735-9390	
	CA	714-370-120		IOMA	319-324-7197 319-354-7371		NC NC	919-832-1551 919-885-0171	L	PA PA	717-233-8531 717-846-3900		WI	414-785-1614	L
	CA	714-662-949		IOMA			WEB			PA .	814-946-8888		MA	608-221-4211	H3
	CA	714-662-849						402-397-0414					WH.	384-522-6261	LJ
	CA	805-324-265		IOMA	515-277-7752		NEB	482-475-8659			809-792-5900 809-833-4535				
	CA	805-486-481		I OWA	515-753-0667 316-265-1241		NEV	782-293-0300							
	S.	007-100-101	1 11	NHT	310-203-1241	п	MEA	702-882-7810	п	ואשטיז	809-840-9110	1			











SAD ICONS

SAD MACINTOSH ICON OCT 26, 1983 Pathi Kenyon

Here is a example:

Power on the Macintosh, holding down the NMI button on the left side of the computer. The sad Macintosh will appear, with a set of numbers under it. This set of code can be divided into two types of code.

OF 000D ~ Sub Codes Class Codes

Class Codes deals with the initial diagnostic code.

1 = ROM test failed meaningless

2 = Memtest - Bus subtest bits set corresponds to suspected bad RAM chips

3 = Memtest - ByteWrite 4 = Memtest - Mod3test

5 = Memtest - Addr uniqueness "

Class Code F = exception, only after diagnostics have passed. This is where the Sub Codes come in.

F = exception

0001 Bus error

0002 address error

0003 illegal instruction

0004 zero divide

0005 check instruction

0006 trapv instruction

0007 privilege violation

0008 trace

0009 line 1010

000A line 1111

000B other exceptions

000C nothing

000D NMI

Another test to see how this works is, remove a RAM chip. Power up the Macintosh.

A new code should appear under the sad Macintosh icon. When I did it, I picked the one closes to the Keyboard connector. The code under the Macintosh was 028000. So number 2's class code tells me that it suspects bad RAM chip. The eight tells me that its the 15th RAM chip.

RAM Chip # Code under Macintosh

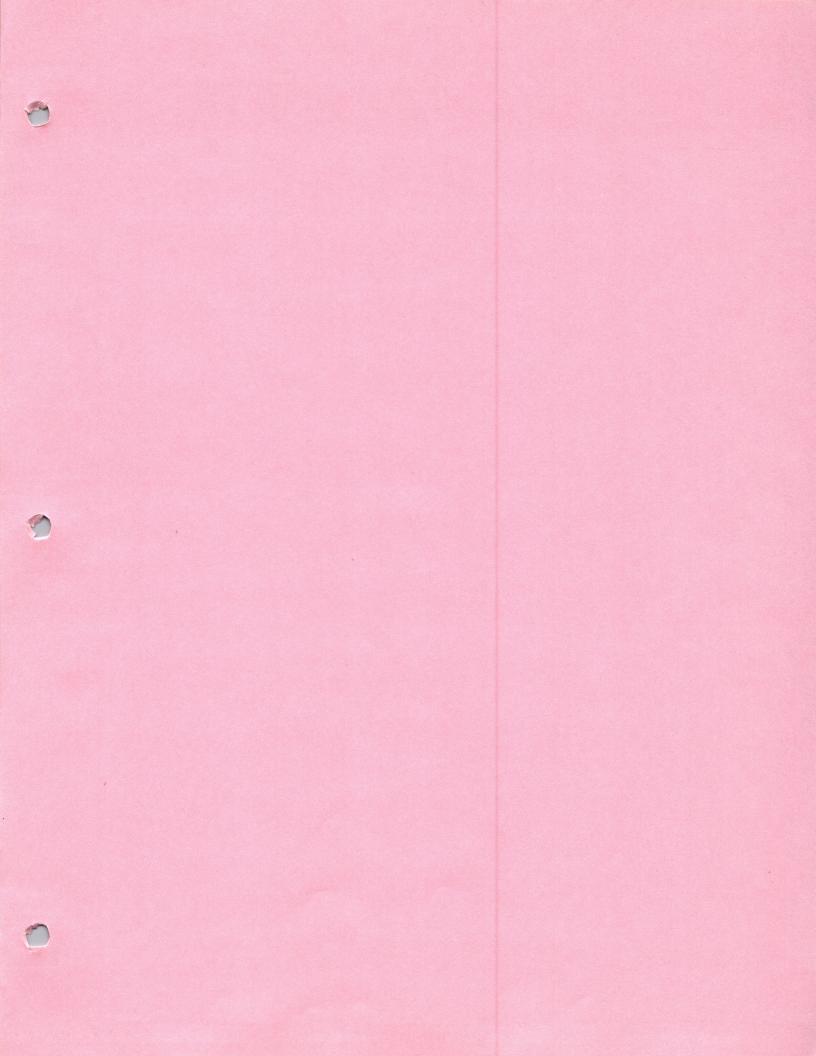
0	=	0001
1	=	0002
2	-	0004
3	-	8000
4	=	0010
5	=	0020

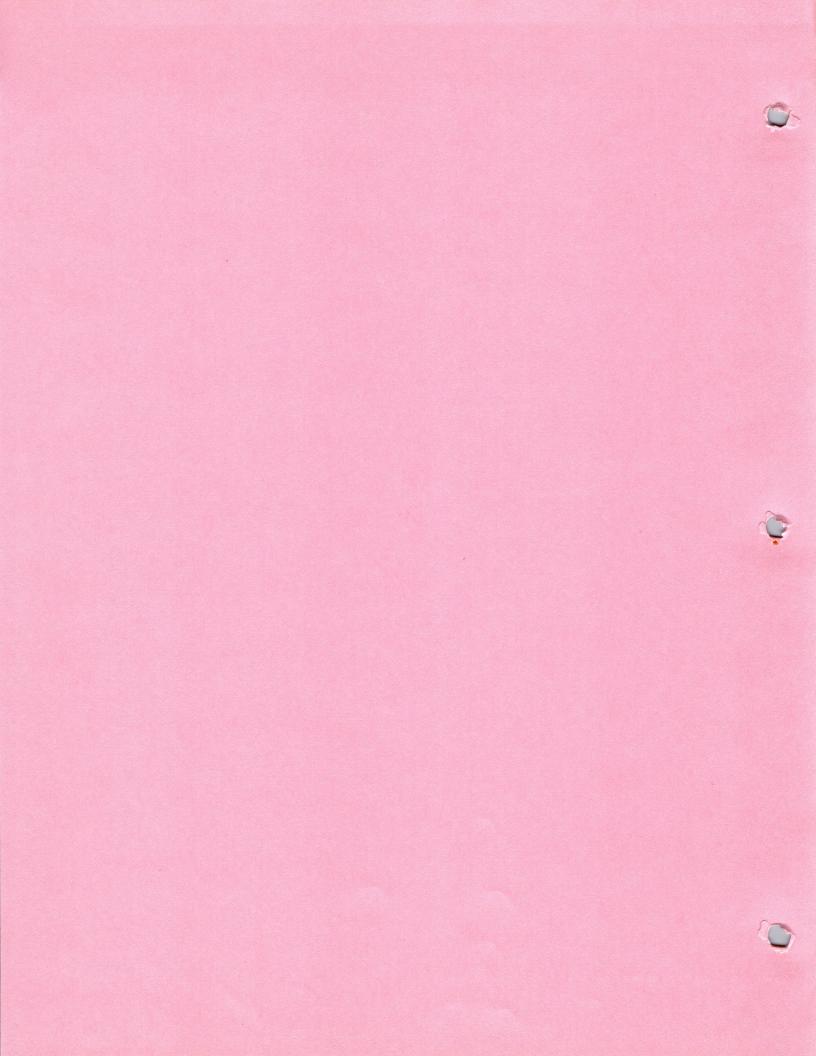
```
0040
 6
 7
                  0080
 8
                  0100
 9
                  0200
                  0400
10
11
                  0800
12
                  1000
13
                  2000
                  4000
14
15
                  8000
```

This is a good example of just one RAM chip being bad, but what is there are multiple RAM chips that are bad? Try taking the 15th and 14th RAM chips out. the Code appears like this. 02C000, we can say since we know the code is in Hex that there are 16 possibilities.

```
0
  0000
   0001
1
2
  0010
3
  0011
   0100
4
5
   0101
   0110
7
   0111
8
  1000
9
   1001
A 1010
  1011
C
   1100 meaning the 15th and 14th chip is bad.
D
   1101
E
  1110
F
   1111
```

This is just a start to understanding what all the codes mean. If we try to keep a list, we should come across all the possibilities.





S(ysmgr - System Manager

C(ontrast - Set the contrast of the screen using < and >.

D(evice - List all mounted devices.

M(ount - Mount a Device

&1 = upper drive &2 = lower drive

&3 = built-in parallel port

&4 = lower connector on parallel card in slot 2
&5 = upper connector on parallel card in slot 2

O(ff - turn machine off

P(rinter - Mount Printer

Serial printer presently is not supported

U(nmount - Unmount a Device

W(ork - Set working Device

It must have a volume named Lisa and all necessary system files. It may be set to the Monitor 12.2 Boot fileware by specifying the drive that it is in.

At that point, all system software will execute off the

fileware.

A(pple - Do not use

F(lip - Do not use

dI(m - Set the delay in seconds before the screen fades. The

count is reset if a key is pressed or the mouse is moved.

L(isa - revision level of your machine

scR(een - Do not use

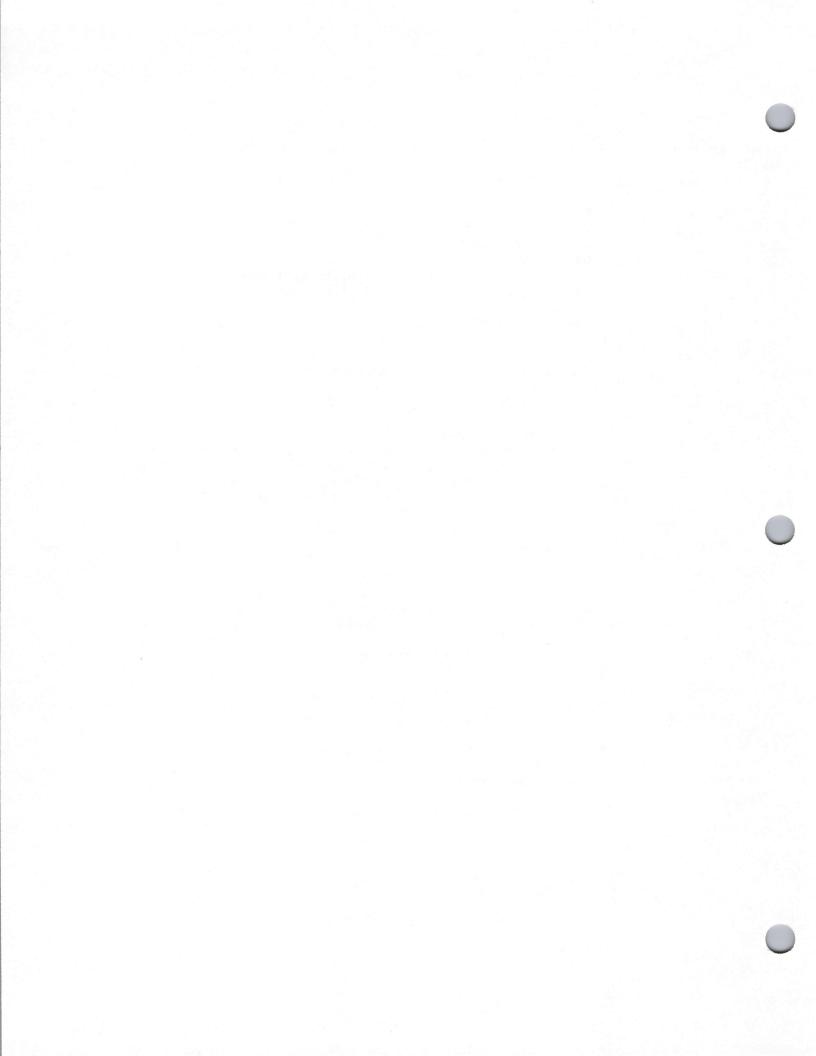
S(et Time - affects the hardware clock

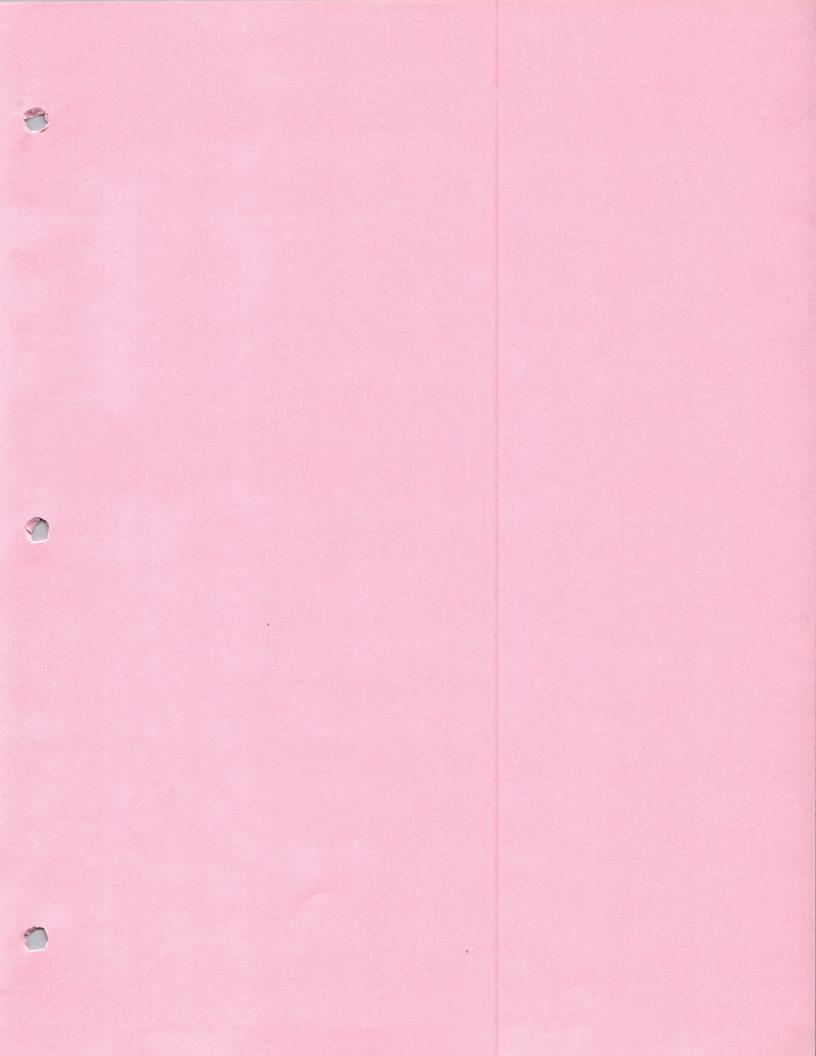
T(ime - displays the time

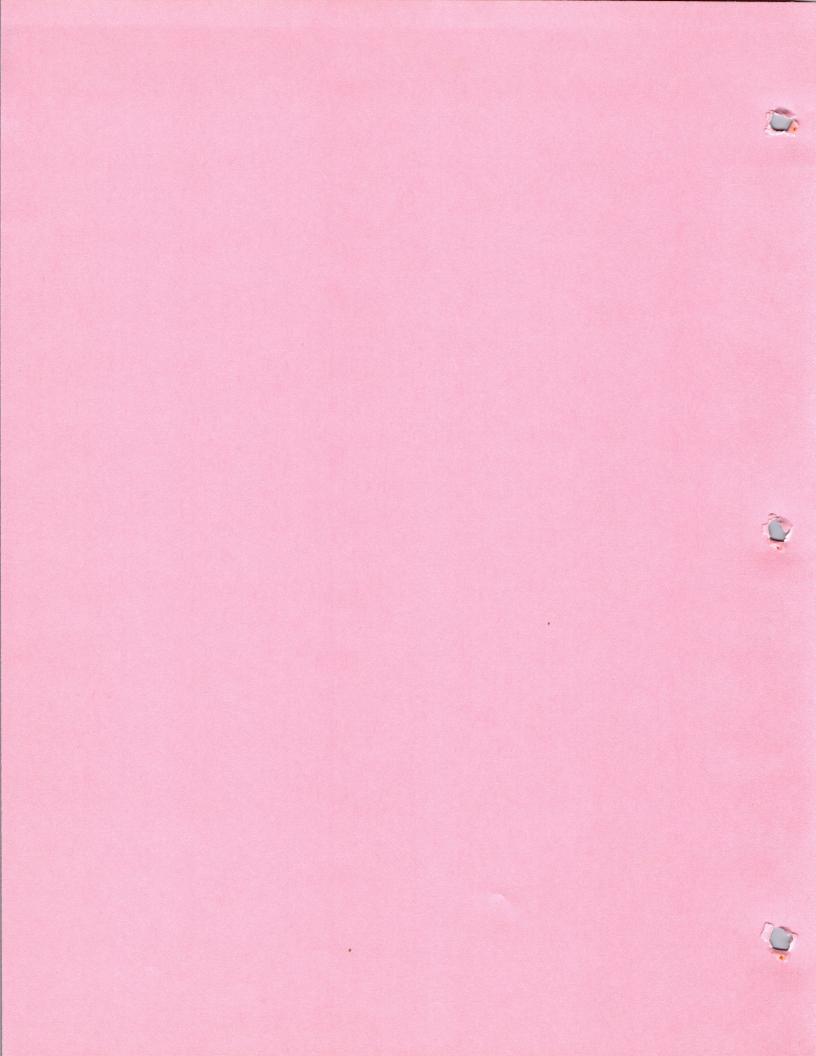
V(olume - set the beep volume

Z(ero - remove all volumes on a device

Q(uit







Monitor filename syntax and file structure:

File containers are called devices. A disk drive and a ProFile are each 1 Device. All of the devices that the system is aware of are listed with the $S(ysmgr\ command\ D(evice.$

Each Device has a name which ends in a slash '/'. Each has a number associated with it which is prefixed by a and-sign '&'.

&l = upper drive

&2 = lower drive

&3 = built-in parallel port

&4 = lower connector on parallel card in slot 2

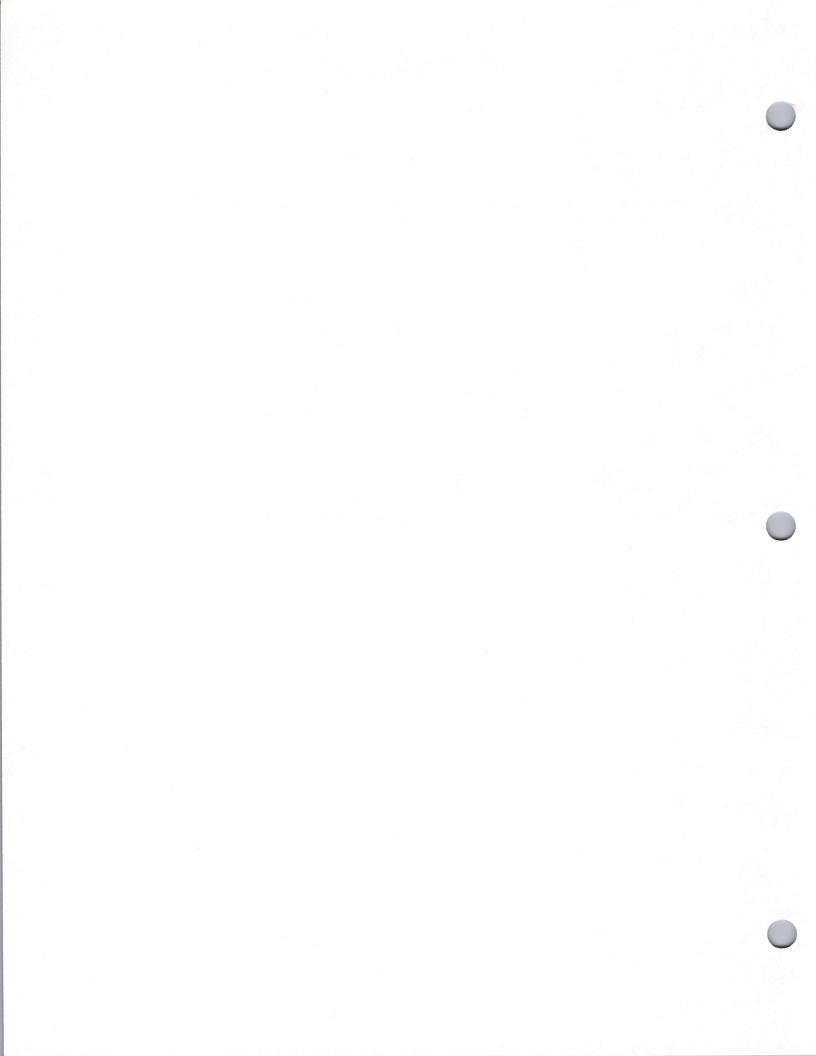
&5 = upper connector on parallel card in slot 2

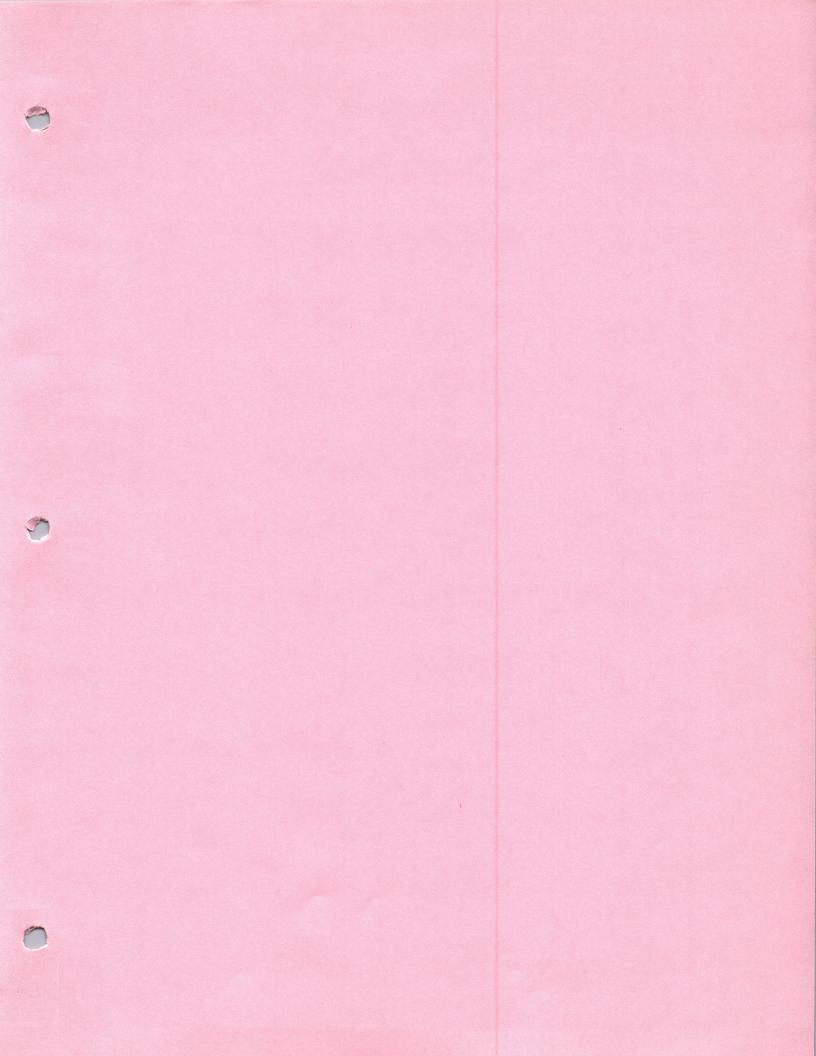
The device contains one or more volumes. The volumes on a device are listed with the $F(ilers\ vM(inger\ command\ L(ist.\ All\ volumes\ known\ to\ the\ system\ can be listed with the <math>F(iler\ command\ O(nline.\ vM(inger\ command\ o(nline.\ o(nline.\ vM(inger\ command\ o(nline.\ o(nline.\$

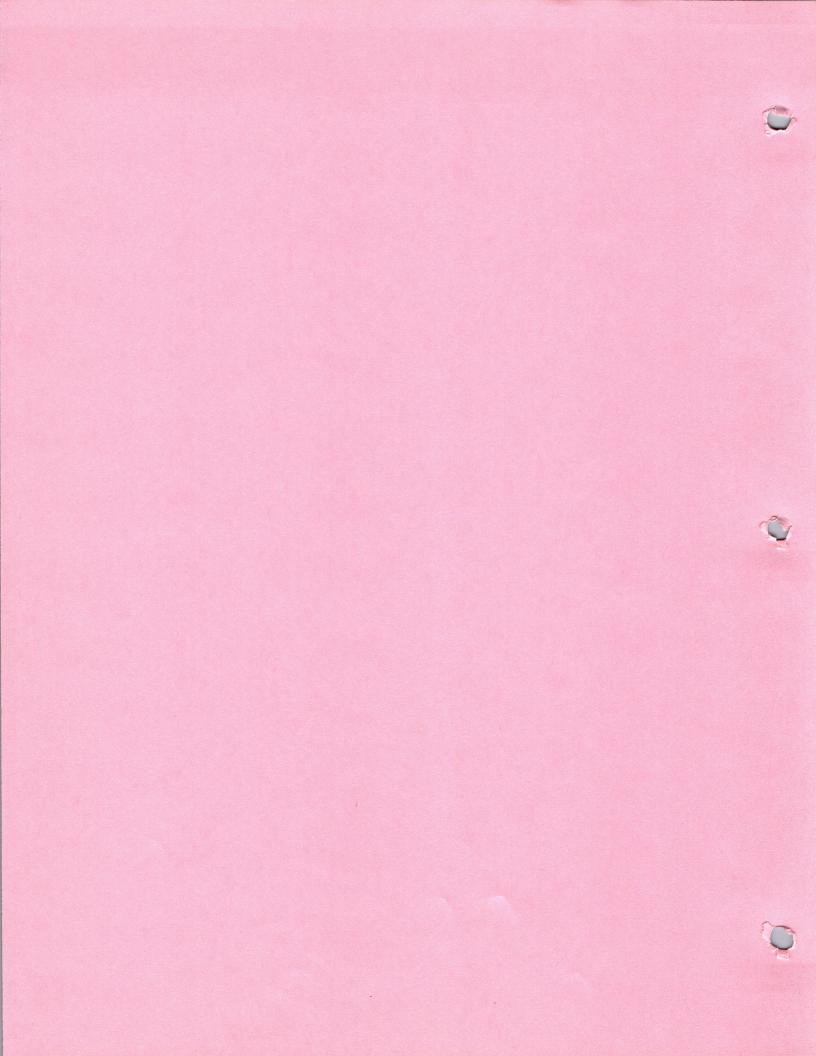
Each Volume has a name which ens in a colon ':'. Each has a number associated with it which is prefixed by a pound-sign $'\sharp'$.

#5 = the primary volume
Disk volumes may be #4 (special), #5, #13..#20.

Each Volume contains files. A fileware volume can contain 77 files. Each file is stored sequentially on the volume. The space for a new file thus must be sequential. After a large number of new files are made, it is necessary to use the F(iler's K(runch command to regain the fragmented volume space.







ROM 7.0

ROM 7.0 MacsBug Summary

To use MacsBug, include it on your Mac diskette. The system will say 'MacsBug installed' when the diskette is booted. You may also include the Disassembler in the same manner.

The Mac's modem port should be connected to another computer or terminal running at 9600 baud, no parity. Press the interrupt switch after booting the disk. The mouse should freeze and no error message should appear. On the terminal, a register dump should appear, and an asterisk '*' prompt.

Commands available:

DM SM	Display Memory Set Memory	DM 100 100 SM 0 1 2 3 4 5 6	DM RA7,-10 20 SM 0 'ABCDE'
D#	Display/Set data reg		DO 0000FFFF
A#	Display/Set address		AO
SR PC	Display/Set status re		
US	Display/Set program of Display/Set user state		
SS	Display/Set user state Display/Set supervise		
BR	Display/Set break po	ints	BR
	(up to eight)		BR 4DDO 552A
			BR CLEAR
A	Display all address	registers	
D	Display all data reg		
TD	Display all register		
CA	Convert between base		CV \$FDEF
	(all arithmetic is 3	2 611)	CV &65536
	To do hexidecimal add	dition, use CV \$num1,num2	
		btraction, use CV \$numl,-num	m2
	To do hexidecimal neg	gation, use CV \$-numl	
G	Go	continue)	G
		start at 44DO)	G 44D0
		continue until PC = 55EA)	G TILL 55EA
T	Trace		T 17
AT	Trace Traps (traces all traps)	AT
		trace GetNextEvent)	AT 170
		trace all Bit Traps)	AT 158 15F
	(trace GetNextEvent in code	
		block at 5000 - 53FF)	AT 170 5000 53FF
		trace all Bit Traps in code	
		block at 5000 - 53FF)	AT 158 15F 5000 53FF

AB	Break Traps same as AT, but breaks before	re executing trap
HD	Handle Display lists all allocated handles	HD
AD	Data Break	AD 158 15F 5000 53FF
	A simple checksum is calculated for the specific As each Trap is encountered, the checksum is result the checksum differs, the debugger breaks. be made with all four arguments.	ecalculated.
AX	Cancel Break	AX
	Clears the current AT, AB, AH, AC, or HS comman	nd
	Disassembler Calls	
IL	List lists the next 20 instructions	IL 4DDO IL
ID	List One lists one instruction	ID 4DDO
	Debugger Notation	
•	Command Separator Last Address	SM PC 4E71 / G
	(for DM, SM, IL, ID)	DM • 100
,	Offset	DM .,100 100
RA# RD#	Address Register Data Register	DM RA7,-10 20 SM RAO, RD2
	Advanced Debugger Calls	
АН	Heap Break	AH 158 15F
	A heap check is made as each specified Trap is If \$1A3E8 = 0 then the applzone is checked. (If \$1A3E8 <> 0 then SysZone is checked. The trap be greater than \$2E.	(default)
	An error returns: Bad Heap at Al A2 Al = the previous block pointer A2 = the bad block pointer	where:

This call checks the heap as described in AH. An error is returned if any of the following conditions are true:

The block size is past the top of memory

The block size is odd

For tree blocks, the next link is negative or past the top of memory

For tree blocks, the previous link is negative or past the top of memory

For rel. blocks, the back pointer is odd

The heap base + back pointer is past the top of memory

The heap base + back pointer do not point to the right master pointer.

HS Heap Scramble

HS

If the traps NewPtr, SetPtrSize, NewHandle, SetHandleSize, HandleZone or ReAllocHandle are encountered, the heap is scrambled before executing the trap. It also preforms a heap check before scrambling on all traps > 30.

MR Magic Return

MR

This assumes the first word on the stack is a return address generated by a BSR or JSR. It substitutes a break point for the return address. The execution continues until a break occurs. Then, SR is restored.

This is not nestable. All other break point commands are still active.

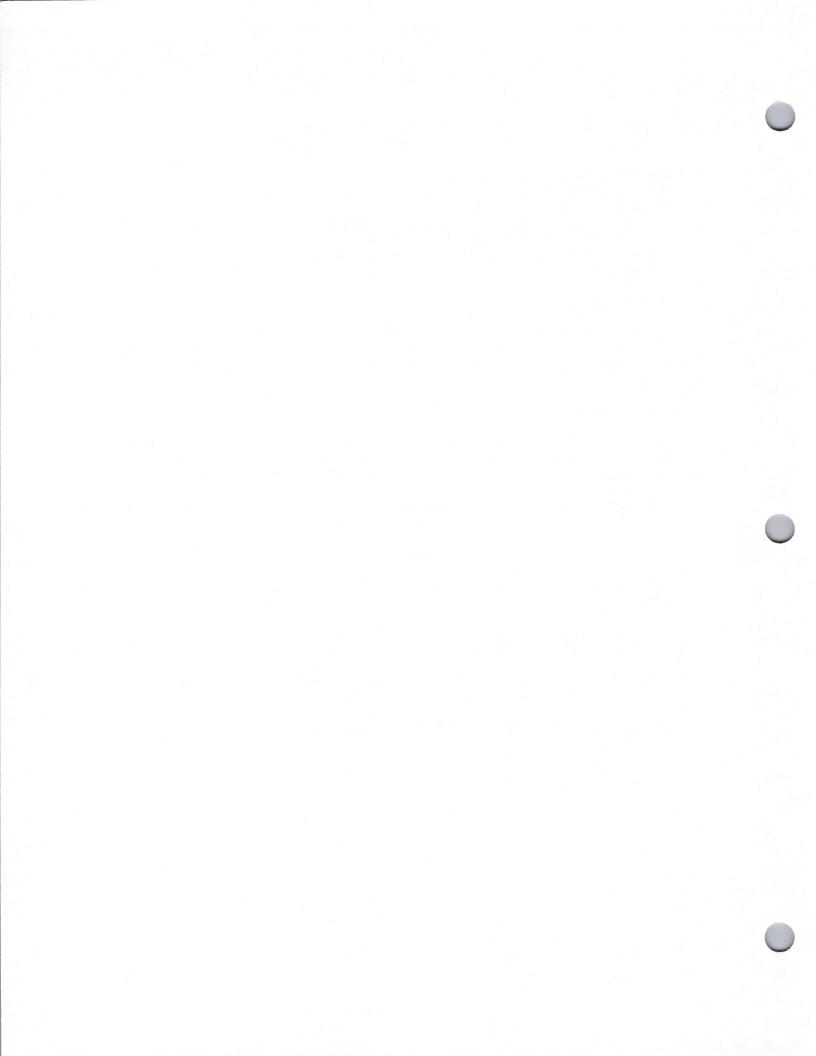
Known Problems

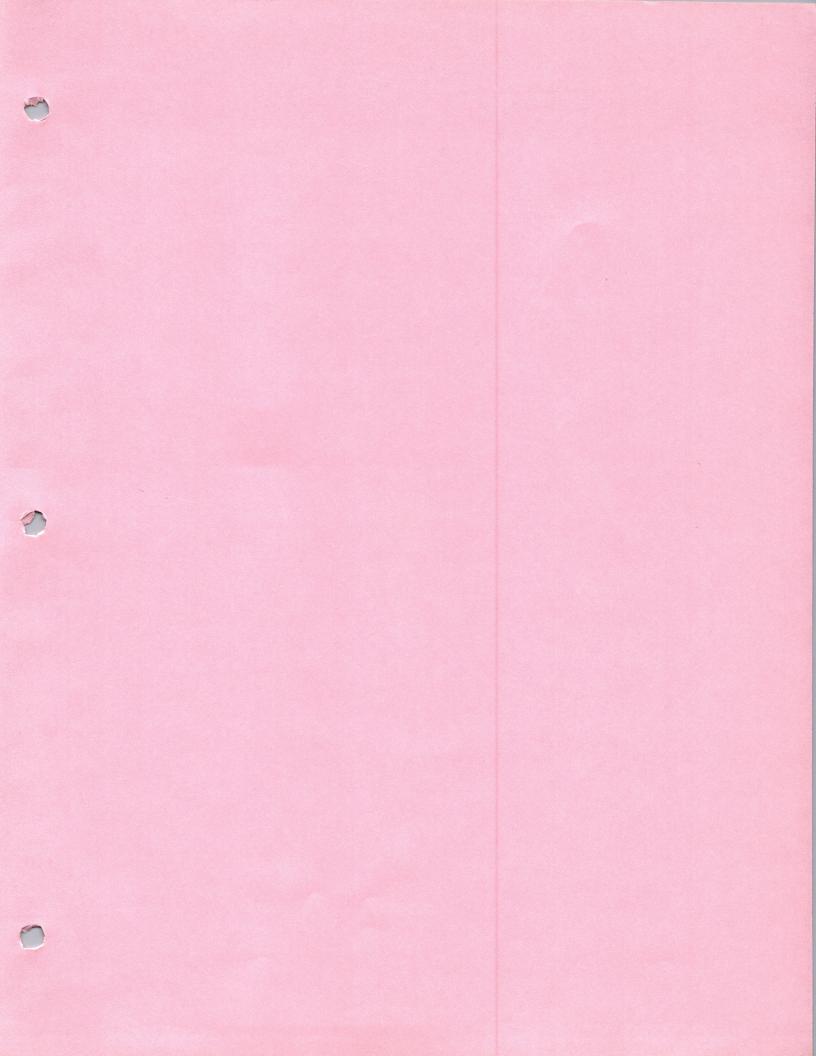
It is a good idea to initialize DM and IL. DM 0, and ID PC, for example.

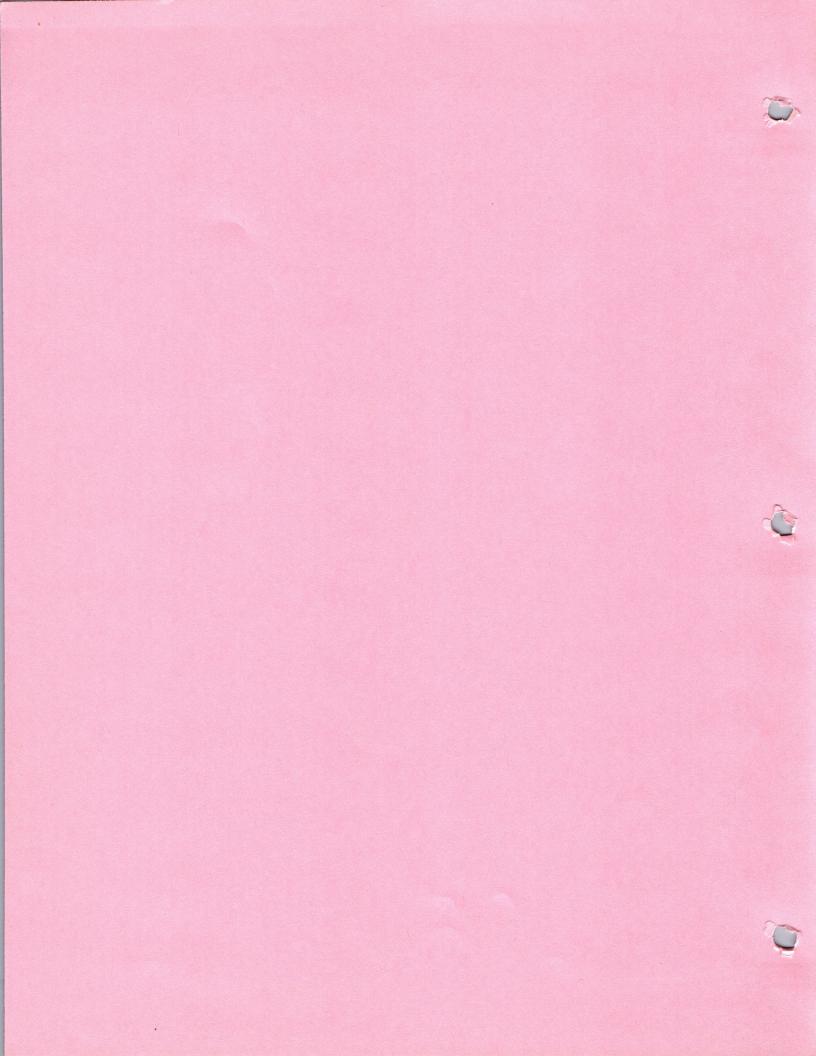
DM RA5, as example, intermittently generates an address error. To fix, explicitly type the address in register.

SM PC 4E71, for example, makes the system respond unreliably if a trap or breakpoint was set at that location.

AT, as example, returns a Line IIII exception. To fix, reboot.







PASCAL DEBUG

Pascal Program Debug Strategy

Use DM to determine where the program is in memory. Seven letters of each procedure and function will appear in the ASCII columns. (The first letter has its high bit set.) The user program usually starts about 4DDO. The mainline procedures and functions are first, followed by the units and external procedures and functions in the order that you link them. Each procedure or function name suceeds the procedure or function code. To make life easier, link your own units before linking ToolTraps, MemTraps and MacPasLib.

If the program doesn't appear to work at all, find the address of the start of the program. It will be immediately after the name of the last procedure or function.

If you disassemble at that address, you will see a LINK instruction for A6 and a LINK instruction for A5. These address registers are used by Pascal to locate all variables and procedural parameters. Global variables are referenced negatively off A5. Local variables are referenced negatively off A6. Procedural parameters are referenced with a positive offset from A6.

ToolBox calls and other calls to unit-resident procedures and functions are made through JSR *+addr instructions. The table ToolTraps is linked to your program, and contains all of the actual calls themselves.

If you are writing a Pascal program that uses the ToolBox, the first thing you probably do is:

InitGraf (@thePort);

This assembles into:

LEA \$FF1E(A5),A0 MOVE.L AO,A7 JSR *+addr

You should see this shortly after the LINK instructions. This establishes the beginning of your program.

To find out where this program fails, interrupt the Finder. Set G TILL addr where addr is the beginning of your program. Restart your program. If the program breaks successfully at that point, continue to use G TILL to selectly execute your program until it fails more spectacularly, or locks up.

You will find that the 68000 code that the Pascal compiler produces is reasonably readable, and that the compiler produces smart code.

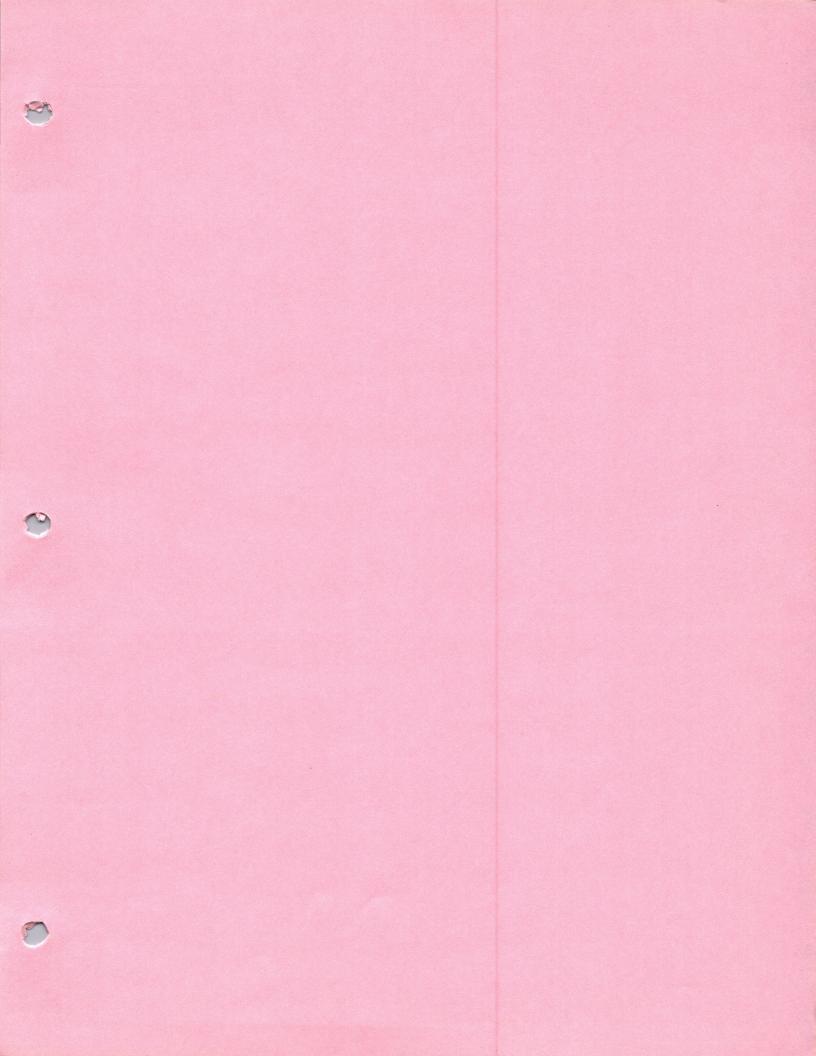
To complement the debugger information, you may want to add debugging code in your program itself. One easy way to do so is to do WRITE s or WRITELN s to the .BOUT port. This information will appear on your debugging terminal. The code fragments required look like:

VAR debug : TEXT;

REWRITE (debug, '.BOUT');

WRITELN (debug, chr(10) {linefeed}, 'This is a test', 12345);

WRITE and WRITELN support strings, chars, packed array of chars, integers, and booleans.



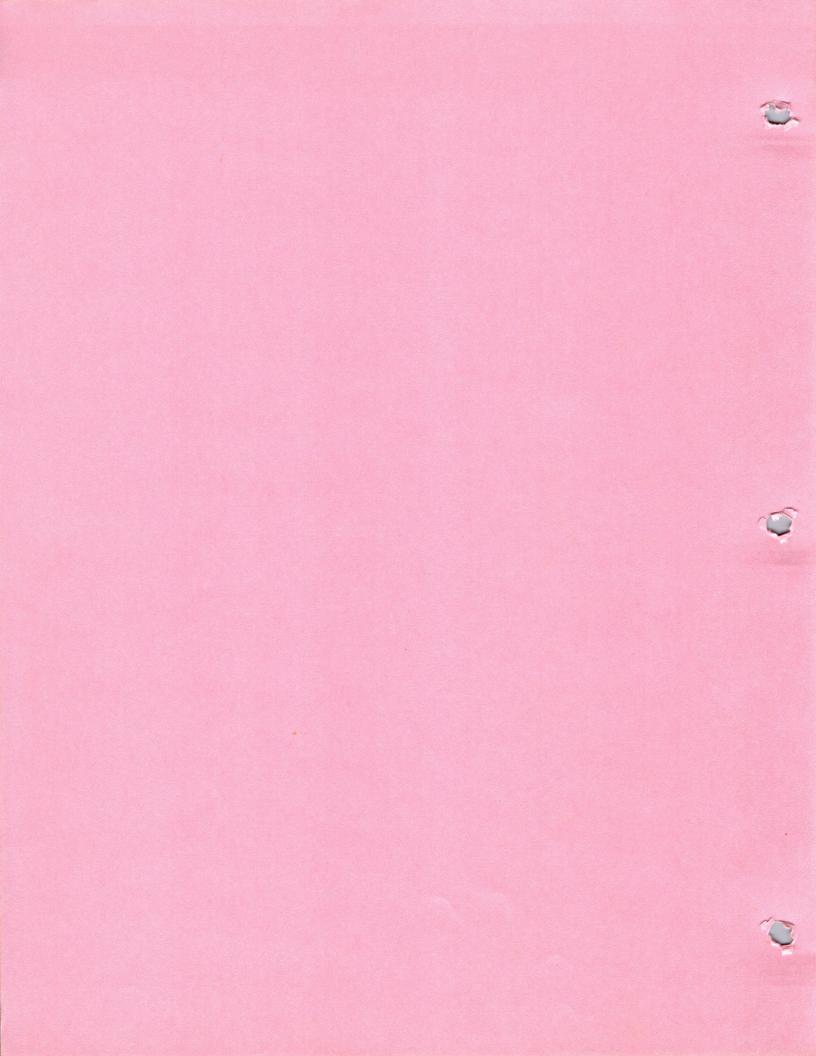


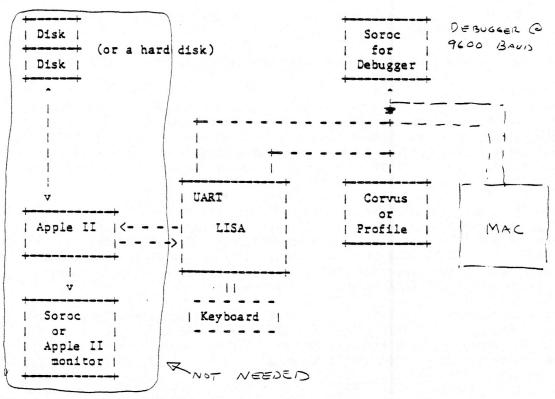
TABLE OF CONTENTS

The	Moni	to	r	•	٠	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	1
The	Pasc	al	C	01	ip i	.1	er														7
The	Link	er				•			•									•			15
The	Asse	mb	le	r		•		•	•		•						•	•			23
Lis	aBug	•					•	•	•			•	•	•						•	37
The	Symb	ol	ic	. [e	u	gge	r		•		•	•		•						47
The	File	r			•			•			•	•								•	53
The	Edit	or	s																		
		Th	e	Li	sa	3	Edi	to	r												59
																					63
Uti	litie	s																			71
	Segme		2 -	4 ,		,	-4	7-					***								
	Segme	TI	Ma	7) (L	ع ء د	rid	1.11		T	15 1	. C	U	III	٠	ıar	ıaş	geo	ier	וב	72
																					74
																					74
	Syste	Co	nf	is	gui	e		•	•			•		•	•	•				•	75 75
																					76
		Ch	20		M.	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	76
																					76
																					76
	File	T-																			
		7;	10	D :	. = .	•															77
		Fi	10	J	i:	'n	:	:	:		:		:	:	:	:		:	:		77 77
	C		- 1	,			L .														
	Source	D 4	6 5	16		Je	oug	81	п	5											78
		E4	77	77	•	•	:	•	•	•	•	•	•	•	•	•	•	•	•	•	80
							s t					•	•	•	•	•	•	•	•		81
													:	:	:	:	:		:	:	
	Objec		. .	1.		-															
	objec				IJ		กศล	8-	115	•											87
		Du	•				•	•	•	•	•	•	•	•	•	•	•	•	•	•	88
								•	•	•	•	•	•	•	•	•	•	•	•	•	89
		OB						:						:	•	•	•	•	•		90
		2.5			-	٠.	•	•	•	•	•	•	•				•	•	•		
		5 V	20	1	-														-	-	9(1)
		GX	ce Re	£			:	:	•	•	•	•	•	•	•	•	•	•	•		90 90

	Hardware Debugging											
	LisaTest	•	•	•	•	0			•	•	•	91
	Performance Measurement	b										
	Perform		0	0	0	0	•	0	0		0	93
	Coverage Analysis											
	Document Compiler Script	•					c		0	e	•	96
	Other Terminal Emulator	0	۰		•	۰			6	•		98
En S	ror Messages					e		0	•		•	101

MONITOR

The Monitor is an operating system for the Lisa computer. Its user interface is patterned after that of the UCSD system on the Apple II. There are several possible system configurations. A standard one is:



The hard disk can be connected directly to the Lisa, or it can be accessed through the Apple II. It can also be omitted.

BOOTING THE MONITOR

To our from a diskette based Apple II, first put the Apple II with the male and diskette in drive #4:. In the female boot in drive #5: and power up the The female boot volume can also reside on a hard disk. SYSTEM.STARTUP on the boot volume automatically executes MONBOOT, the proper a that starts up a Manitor on the Lisa. If you type space thing the boot process, MONBOOT is no little of the boot process, the debugging version of the little of the booted.

Place the MONITOR 12.2 DISK IN THE UPPER LISAS DIRIVE AND TURN THE SYSTEM ON. NEWER LISAS WILL SHOW A MEMU SELECTION IF YOU PRESS THE COMMAND KEY. BOST ON THE MONITOR DISK.

The Monitor comes up on the Lisa screen. If you want it to appear on the Apple II monitor or the Soroc connected to the UART port, change the MON.STARTUP program as follows:

- 1) for the Apple window: remove MON.STARTUP
- 2) for the Lisa window (the default): transfer MONSTARTI.OBJ to MON.STARTUP
- 3) for the UART window: transfer MONSTART2.OBJ to MON.STARTUP

To move the Monitor around after booting, execute MOVESOROC. MOVESOROC simply asks you for the new source and destination for Monitor I/O: A(pple, L(isa, or U(art. Input always comes from the terminal to which output has been directed. WRITELNs used for debugging purposes appear on the Monitor screen, so you may not always want the application and Monitor screens to be together on the Lisa.

CONFIG.DATA tells the monitor how much RAM your system has. The default configuration assumes that you have a megabyte of RAM. For a 256K byte system, CONFIG.DATA should be a copy of NPC4.DATA. For a 512K byte system, use NPC6.DATA. NPC16.DATA is a copy of the default CONFIG.DATA, in case you ever need to back up. See CONFIGURE in the Utilities chapter if you want to change CONFIG.DATA to suit your own needs.

The monitor's keyboard driver supports the Lisa User Interface Keyboard layout. SHIFT is {, SHIFT } is }, SHIFT . is >, and SHIFT , is <.
"-" NMI is the third key from the left in the upper row of the numeric keypad. Garrantly, the "4" key is backspare. The CODE key is in the upper left corner of the keyboard. CODE; is |, CODE + is -, CODE is \,, and CODE " is \. ESCAPE is the upper left key of the numeric keypad. Clear Gontrol-S is the key in the upper right corner of the numeric keypad.

THE COMMAND LINE

The Monitor command line is:

Monitor: E(dit, C(ompile, F(ile, L(ink, A(ssemble, D(ebug, ? [0.1]

There are several hidden commands. Type ? to see them displayed

USE THIS ONE ____ E(dit Lisa-style Editor C(ompile Pascal Compiler I-code generator F(ile Filer THE REAL CO. Intrinsic Unit Sinker (release 8.0 L(ink Linker A(ssemble Assembler Diebug-Symbolic Debugger M(acsBug Lisabug (low level debugger) G(enerate Pascal compiler OBJ file generator ILCSD UCSD Editor X(ecute Execute a program or an EXEC file

The Monter recognizes male volumes and logs them off-line so that they cannot be accidentally overwritten. The volume MEMORY: is always available. MEMORY: allows you to use the Lisa PAN as a file storage area. Of course, anything in MEMORY: is loss then the power is turned off, or the system is rebooted. MEMORY: is mounted as unit #4:, and its default size is 10 blocks. Its size can be changed by the Z(ero command in the Filer, or by the CHANGEMEM program described in the Utilities section this annual.

When X(ecuting a program, the monitor searches for the program filename as follows:

Filename
Filename.OBJ
*Filename.OBJ
Filename.TEXT (* as an exec file *)

When you invoke a program from the Monitor command line (F for Filer.Obj, for example), the Monitor looks first at the MEMORY: volume.

EXEC FILES

EXEC files can be used on the Monitor. They must be created in the Editor (there is no M(ake command). To execute such a file.

X(ecute <filename>

If an object file exists with the same name as that of the EXEC file, the object file is executed. The first character of an EXEC file (a textfile) defines the termination character. The first occurrence of two terminators marks the end of the EXEC file. Certain portions of the system (the compiler, for example) terminate an EXEC file if an error is encountered. If you X(ecute a .TEXT file, the monitor assumes that the file is an EXEC file. EXEC files cannot be nested, nor can parameters be passed to them.

LOW MEMORY LAYOUT

Applicable

From	- To.	Description
	OOFF	Exception vectors (see 68000 manual)
0000		Memory configuration map (see page ##)
0100	Olff	Free space for user assembly globals
0200	0300	KCS numerics status information
0300	0341	
0342	OSFF	Free space
0400	OFF	LisaBug Globals
0800	08FF	Boot stack
0900	OAFF	LisaGraf Globals
0800	OBxx	Unit Table
0000	OCFF	Pointer array
ODOO	ODxx	Syscom, miscinfo
0E00	oef f	String buffer
OFOO	OFFF	Unused (reserved) space
1000	17FF	User code buffer
1800	3FFF	User Jump Table
4000		Heap Bottom
Registers		Description
A7		Stack Pointer
A6		Stack frame Pointer
A5		Global Data Pointer
A3-A4		Used for code optimization
A2-A0		Scratch
n4-n0		
\ DO-D3		Scratch
D4-D7		Used for code optimization

Registers D3 and A2 may someday be used by the compiler for code optimization.

SEE NOTE ON

NOW G (ENERATE

OBJ FILE GENERATOR

Apple Computer Inc. POS DEVELOPMENT SYSTEMS CROUP

OCT.4

TO: Macintosh group Fred Forsman 2-P Revised version

FROM: Al Hoffman &.

SUBJECT: New Pascal code generator for Macintosh

The Lisa Pascal code generator has recently been updated with two new options. These options are activated by typing them on the "Input file?" command line instead of the input file. The code generator will then re-prompt for the input file.

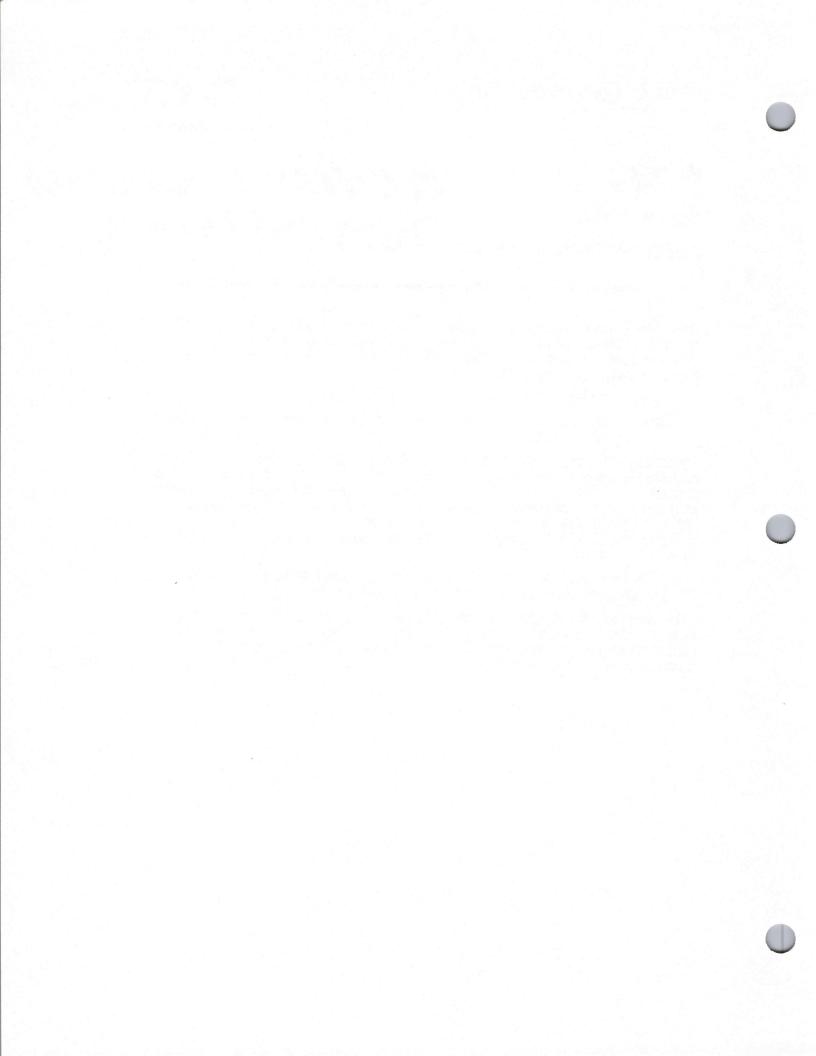
\$A+ Preserve registers A2 and D3 across procedure calls.

\$M+ Perform Macintosh code generation. Automatically invokes \$A+ when used.

The Macintosh option performs three things. First, it inserts code, if necessary, in each procedure to preserve registers A2 and D3 across the procedure. This action is also activated by \$A+. Secondly, the LINK A5,#n statement at the beginning of main programs is forced to LINK A5,#0. Thirdly, the TST.W instruction at the beginning of procedures and the TST.B instruction at the beginning of references to string parameters is forced to be eliminated.

When \$M+ is activated, the code generation process ends by displaying "MACINTOSH code generated." at the bottom of the display.

This version of the code generator is available now from Andy Hertzfeld or myself. It is compatible with (at least) version 12.2 of the Monitor. Earlier versions of the code generator cannot guarantee code that is compatible with the Macintosh ROM.



MEMORY MAP

\$0

A set of very detailed memory maps can be found in the Linker chapter. We give below a general view of memory and a detailed view of the Monitor's Map Table.

Memory <	UART \$11
I/O <	
Screen Memory	+ Memory Top \$11
	+
LisaBug	+ Buffer Pointer \$10
Disassembler Graphics	Not Used \$10
Monitor	Monitor Top \$10
Code <	Monitor Bottom \$10
Stack	Default Stack Pointer
•	Assembly Globals
Heap	Map Table at \$100
MEMORY:	1

THE MAP TABLE

Monitor bottom	\$100
Monitor top	\$104
	\$108
Buffer pointer	\$10C
Screen base	\$110
Memory top	\$114
Port to Apple II	\$118
UART	SIIC
Ptr to Lisabug jump table	\$120
Ptr to GOTOXY	\$124
Ptr to Soroc driver	\$128
Ptr to soft break table	\$12C
Ptr to UART driver	\$130
Ptr to Corvus card	\$134
Ptr to base of heap	\$138
	\$13C
Ptr to user's last A6	\$140
Ptr to MEMORY:	\$144
Ptr to Twiggy driver	\$148
Ptr to hard disk Jump Table	\$14C
Ptr to debug card	\$150
Ptr to loader for IUs	\$154
	\$158
	\$15C
	\$160
	\$164
Apple net	\$168
Apple net	\$16C

Many of these vectors can be changed by the CONFIGURE program described in the Utility section of this manual. The main vector of interest is the default stack pointer (\$13C). The utility program SETSP can be used to change the default stack pointer value temporarily. CONFIGURE can be used to change it permanently. Unused addresses are reserved for future use by the Monitor.

THE COMPILER

Files needed: COMPILER.OBJ

CODE. OBJ

MPASLIB.OBJ, NOFPLIB.OBJ, or IUPASLIB.OBJ

GENERAL INFORMATION

The compiler is split into two programs, COMPILER.OBJ and CODE.OBJ. COMPILER.OBJ (invoked by the Monitor's C(ompile option) parses the Pascal program text into semantically equivalent tree structures. CODE.OBJ (invoked by the G(enerate command) then turns these trees into 68000 code. The compiler follows the proposed ISO standard Pascal with some exceptions and extensions. A complete definition of Lisa Pascal can be found in the Pascal Language Reference Manual. The definition of I-code formats and MPaslib information can be found in the Development System Internal Documentation.

The compiler first asks for the

Input file -

The .TEXT extension is added, if necessary. In the following prompts, the bracketed text is used if you leave out that portion of the file names.

Listing file (<cr> for none) Output file [<input name>] [.I]
Debug file [<input name>] [.DBG] -

If you do not want a debug file created, type (ESC) (cr). (cr) always accepts the default setting. If you write both the .I file and the .DBG file to the same volume, use the [*] specification on the .I file to avoid space problems. The trouble arises when you have one large block on the volume. When the operating system allocates space for a file, it gives the file all of the largest block it can find unless you specify otherwise. If no other block of space exists and all of the existing block has been allocated to the .I file, you get a "no room on vol" error when the system attempts to open the .DBG file, even if there is plenty of room for both. The [*] specification tells the operating system to allocate only half of the largest available block to the file.

The Pascal run time support routines are in MPASLIB. If you do not need the floating point arithmetic routines, you can use NOFPLIB instead of MPASLIB. If you are using intrinsic units, use IUPASLIB.

COMPILER OPTIONS

\$D+ or \$D- - If the \$D option is on (the default), the compiler places procedure names in the object file. The object file is slightly larger, but LisaBug use becomes much more pleasant.

SDECL - Compile time variable declaration (conditional compilation). Compile time variables must be declared before they can be used (in SSETC), and all declarations must precede the first procedure or function definition in the program. The SDECL compiler option does not exist until the version 8.0 compiler.

SE filename - Starts logging compile time errors as they are encountered.

This option is analogous to the \$L option.

SELSEC - Conditional compilation.

SENDC - Conditional compilation.

SI filename - Includes the file 'filename' in the compilation. The filename cannot begin with a '+' or a '-'.

SIFC - Conditional compilation.

SL filename - Starts making a listing of the compilation in file
'filename'. If a listing is already in progress, that
file is closed and saved before the new listing file
is opened.

\$L+++ or \$L--- - The first +/- turns listing on (+) or off (-) during the first pass. The second +/-, if present, turns on or off the listing with object code offsets during the second pass. The third +/-, if present, controls the production of an interlisting during the second pass.

SS segment - Starts putting code modules into the segment named 'segment'. The default segment (' ') holds the main program and all built-in support code. All other code can be placed in any segment.

SSETC - Compile time variable declaration and assignment.

\$U filename - Searches the file 'filename' for any subsequent units.

\$X+ or \$X- - Turns stack expansion code on (+) or off. The default
is \$X+.

\$%% - Allows the use of percent signs as legal characters in identifier names. The default is \$%% -. The % option should not be used by normal applications.

PACKING INFORMATION

Packed records are very expensive in terms of the number of bytes of code generated by the compiler to reference a field of a packed record. In general, you should avoid packing records unless there are many more instances of a particular record than there are references to it.

Packed arrays are also code-expensive, with one exception. Packed arrays of char are treated as a special case, and the code associated with them is compact.

To paraphrase von Neumann, anyone who needs to know the details of the packing algorithms is in a state of sin, but the following is provided for the sake of completeness.

Elements of packed arrays are stored with multiple values per byte whenever more than one value can be fit into a byte. This only happens when the values require 4 bits or less. Values requiring 3 bits are stored into 4 bits.

The first value in a packed array is stored in the lowest numbered bit position of the lowest addressed (most significant) byte. Subsequent values are stored in the next available higher numbered bit positions within that byte. When the first byte is full, the same positions are used in the next higher addressed byte. Consider the following examples:

a: PACKED ARRAY[1..12] OF BOOLEAN

byte	1:						bic	0
a8	a7	a6	a5	a4	a3	a2	al	
byte	2:							
	- Unus	ed	-	al2	alli	a10	a 9	1

```
b: PACKED ARRAY[3..8] OF 0..3
    byte 1:
                 a[5] |
                           a[4] |
                                     a[3]
        a[6] |
    byte 2:
        --- Unused --- | a[8]
                                  1 a[7]
c: PACKED ARRAY[0..2] OF 0..7
    PACKED ARRAY[0..2] OF 0..15
    byte 1:
             a[1]
                                a[0]
    byte 2:
      -- Unused --
                             a[2]
```

You can use the @ operator to poke around inside any packed value and thereby discover what the packing algorithm (probably) is. For example, to get the data given above, you can use a program like the following:

Consider also the following program fragment:

```
BITE = 0..255;
WORDSWAP = PACKED RECORD
                CASE INTEGER OF
                    0: (HWord: INTEGER);
                    1:(HiByte,LoByte:BITE);
                    2:(High:BITE;
                        Low:BITE);
                    3:(Hex1, Hex2, Hex3, Hex4:0..15);
                    4:(Bool:0..1;
                        Oct1, Oct2, Oct3, Oct4, Oct5: 0...7);
                    5:(Al:0..15;
                        B1:0..7;
                       B2:0..7;
                       B3:0..7;
                        B4:0..7);
                    6: (Bin: PACKED ARRAY[0..15] OF 0..1)
                 END;
```

Each variant gets packed into 16 bits. The question then is, where in the 16 bits do the various portions of the variants get placed:

115	Integer
HiByte LoByte	HiByte, LoByte: 0255
High Low	High: 0255; Low: 0255;
Hex1 Hex2 Hex3 Hex4	Hexm:015
B Oct1 Oct2 Oct3 Oct4 Oct5	B:01; Octn:07;
Al	Al:015; Bn:07;
17 6 5 4 3 2 1 0 F E D C B A 9 8	Variant #6 (using hex digits)

LISA PASCAL AND APPLE PASCAL

Lisa and Apple Pascal are quite similar. We give below a list of the major differences, and a section of hints for translation from Apple to Lisa Pascal. Full details can be found in the Pascal Language Reference Manual.

EXTENSIONS TO APPLE PASCAL

@ Operator
CASE OTHERWISE Clause
POINTER function
Hexadecimal constants
DISPOSE
ORD4 function
Global GOTO
Parametric Procedures and Functions

DELETIONS FROM APPLE PASCAL

Initialization block in UNIT declaration
PWROFTEN, TREESEARCH, BYTESTREAM, WORDSTREAM, KEYBOARD
Extended comparisons
Some Compiler options
SEGMENT Procedures and Functions

REPLACEMENTS FOR APPLE PASCAL FEATURES

Long Integers -- 32 bit integers
Scan -- ScanEq and ScanNe
TURTLEGRAPHICS and APPLESTUFF -- LisaGraf

TRANSLATION FROM APPLE PASCAL TO LISA PASCAL

Translation of Apple Pascal programs is usually not very difficult. The following hints may be of use to you if you find yourself saddled with the translation task. Thanks to Ken Friedenbach for the hints!

```
MOVELEFT(Source_Buf[i], Dest_Buf[k], n) can be translated into:
```

FOR LocalI:=0 TO n-1 DO Dest Buf[LocalI+k]:=Source_Buf[LocalI+i];

It may be necessary to declare the local integer used as the FOR loop control variable.

MOVERIGHT(Source_Buf[i],Dest_Buf[k],n) becomes:

FOR LocalI:=n-1 DOWNTO 0 DO Dest Buf[k+LocalI]:=Source Buf[i+LocalI];

```
FILLCHAR(Buf[i],n,Ch) becomes:
    FOR LocalI:=0 TO n-1 DO Buf[i+LocalI]:=ch;
i:=SCAN(n, <>ch, Buf[k]) becomes:
    LocalI:=0;
    IF n>0 THEN
      WHILE (LocalI<n) AND (Buf[k+LocalI]=ch) DO LocalI:=LocalI+1
      WHILE (LocalI)n) AND (Buf[k+LocalI]=ch) DO LocalI:=LocalI-1;
    i:=LocalI;
    If SCAN is looking for =ch, just substitute <>ch in the loops above.
READ(KEYBOARD, ch) becomes:
    UNITREAD(2, Charr, 1);
    ch:=ChArr[0];
    where charr=packed array [0..1] of char.
EOLN(KEYBOARD)
    can check the character read above. If ch=CHR(13) then EOLN is true.
KEYPRESS
    is NOT UNITBUSY(2).
Strings must be given a length, non-local EXITs must be replaced with GOTOs.
ClearScreen and other such functions can be handled by Jim Merritt's
CUSTOMIO unit. ClearScreen on the Lisa is presently WRITE(CHR(27),CHR(42));
If underbars are used in the Apple Pascal program, they must be used
consistently (they are ignored by the Apple Pascal Compiler!).
If the Apple Pascal units have code in the intialization block, put it
in a procedure called at the beginning of the program.
```

To force segments to be resident, build a chain of dummy procedure calls that forces the loader to keep them all in core. The main program then becomes a procedure called by the top of the chain. Say we have 3 segments called SEG1, SEG2, and SEG3, and have put our main program into a procedure named MAIN PROGRAM. We can now force everything to be memory resident by adding the following procedures:

```
(*$S SEG1*)
Procedure Kludge3;
BEGIN
Main_Program;
END;
(*$$ SEG2*)
Procedure Kludge2;
BEGIN
Kludge3;
END;
(*$S SEG3*)
Procedure Kludgel;
BEGIN
Kludge2;
END;
                *)
(*$$
BEGIN
Kludgel;
                (* end of main program *)
END.
```

THE LINKER

Files needed: LINKER.OBJ or IULINKER.OBJ

GENERAL INFORMATION

The Linker combines object files. Its input consists of commands and object files. Its output consists of object files, link-map information, and error messages. Partial links are allowed. The output of the compiler must be linked with some version of PASLIB.OBJ before it can be executed. Other object files, including libraries, partial links, and object files produced by the Assembler, can also be linked into the output object file.

The Intrinsic Unit Linker (IULINKER.OBJ) expects to find the file *INTRINSIC.LIB even if you are not using any intrinsic units. LINKER.OBJ ('The Linker' in this chapter) expects to find LOADER.IMAGE somewhere on the system.

LINKER PROMPTS

The linker first prompts for the names of the input files:

Input file [.OBJ] -

It continues to ask for input files until you type (cr>. The next request is for the

Listing file -

Type <cr> if you don't want any listing file. The last request is for the name of the

Output file [.OBJ] -

LINKER COMMAND FILES

The Linker can read commands from a text file. At any time you can switch to such a file by typing '<' followed by the name of the file in which the commands reside. If there is a blank line in the file, the Linker assumes that this line is equivalent to the <cr>
typed to end input file input. The line after the blank line (if any) is the listing file name, and the line after that is the output file name. These two files need not be given in the command file.

LINKER OPTIONS

Linker options can be entered at any time in response to the prompt for an input file. The options do not have any effect until the link begins. In particular, segment names cannot be mapped to several different names.

The Intrinsic Unit Linker has the following options:

- +A Alphabetical listing of symbols. The default is -A.
- +D Debug information. The default is -D.
- +H num +H sets the maximum amount of heap space the Operating System can give a program before allowing it to die. Here, as in the other options, 'num' can be either decimal or hexadecimal.
- -H num -H sets the minimum amount of heap space needed by a program.
- Location ordered listing of symbols. The default is -L. The location is the segment name plus offset.

+M fromName toName

+M maps all occurrences of the segment 'fromName' to the segment 'toName'. This allows you to map several small segments into a single larger segment. You can thereby postpone the segmentation decision until link time by using many segment names in the source code.

- +P Production link. The default is -P. +P produces a 'production' .OBJ file. A production object file does not contain information used by the debugger and the linker, and intrinsic unit files do not contain a jump table. The production object file can be executed, but cannot be handled by the linker or the debugger.
- +3 num

+S sets the starting dynamic stack size to 'num'. The default is currently 10000.

+T num

+T sets the maximum allowed location of the top of the stack to 'num'. The default is 128K.

? Prints the options available and their current values.

The Linker has the following options:

- ? Print out the options and their current values
- Q Use Quick_Load blocks in place of Executable blocks. The Monitor has never supported this option.

Do a Physical (+P) or Logical (-P) link. +P is the default. The logical link uses the MMU's to map logical addresses into physical memory. 'The physical link maps all of memory linearly. A logically linked program is more sensitive to uninitialized pointer problems than a physically linked program. If a physical link is performed, the linker and the executable program it produced must execute with the default stack pointer set to the same location. The default stack pointer value is \$80000.

THE LINKER OUTPUT FILE

If no errors occur during the link, the output file contains the result of the link. If all external references are resolved and a starting location is specified, the output file is an executable object file. You must link in MPASLIB.OBJ or its equivalent to resolve all external references.

ERROR MESSAGES

The Linker reacts in three general ways to dubious usage. It gives a warning message if some action cannot be performed. This kind of message can be distinguished from the others by carefully noting that it begins with:

*** Warning

In order to recover from the error, simply reenter the command correctly, and all will proceed as though no error had occurred.

An error that makes it impossible for the Linker to complete the link successfully causes a message that begins:

*** Error

The link process can be continued, however, so that any further problems can be discovered.

A fatal error causes the link to be terminated immediately and sends a message beginning with:

*** Fatal Error

See the section on errors for a complete list of the Linker error messages.

EXTERNAL NAMES

An external name is a symbolic entry point into an object module. All such names are visible at all times—there is no notion of the nesting level of an external name. External names can be either global or local. A local name begins with a \$ followed by 1 to 7 digits. No other characters are allowed. A global name is any name which is not a local name.

The scope of a global name is the entire program being linked. Unsatisfied references to global names are allowed. Only one definition of a given global name may occur in a given link.

The scope of the local name is limited to the file in which it resides. When a partial link is done, global names are passed through to the output file unmodified, but local names are renamed so that no conflicts occur between local names defined in more than one file. All references to a given local name must occur within the same input file.

MODULE INCLUSION

The first file presented to the Intrinsic Unit Linker must be either a main program file to be linked, or an unlinked intrinsic unit file. You cannot have both intrinsic unit and main program files in a single link. All modules from a non-library file are included in the output file. Only those modules which are needed in the link, however, are taken from a library file. The Linker considers a module to be needed if:

- 1) it defines an unresolved global name, or
- 2) it is referenced by a module in the same library file that is included in the output file.

A module is not included simply because it references an already defined global name. Thus, the inclusion of a library module is dependent on the order in which files are specified to the Linker—the module must be specified after the modules that reference it. You can easily use an alternate module to one in a library by including the alternate prior to specifying the library file.

After linking an intrinsic object file and before referring to it in another link, ypou must update the segment and unit tables in *INTRINSIC.LIB with the IUMANAGER utility. IUMANAGER is described in the Utilities Chapter of this manual.

STRUCTURE OF AN EXECUTING PROGRAM

When a program is executing, the Lisa memory map is:

	L Sames Manage		(Top of memory)
	Screen Memory		10) (Screen Base)
	Bootfiles (LisaBug, GotoXY) (Core, MonSoroc)	1	
	Monitor	+< (\$1) 	04) (Monitor top)
CODE	Program Code	+< (\$10 	00) (Monitor bottom)
JUMP	(Jump Table) (data ptrs)	(see below)	(physical link)
TABLE	pars to main prog	+< (\$1. (see below)	3C) (SETSP default)
	Globals		5) (top of globals)
	Stack (grows down)		
DATA	(locals)	+< (A	6) (stack frame pointer)
	+-	+< (A	7) (top of stack)
	Heap (grows up)	İ	
• • • •	MEMORY:	+ -	
	Monitor heap	i.	
	Jump Table	+< (\$1. +< \$180	(logical link)
	Monitor globals		
	Exception vectors	+< 501	00
		+< \$00	00

A physical link places the Jump Table above A5, but a logical or Intrinsic Unit link places the Jump Table at \$1800 to free up A4 for code optimization. Even when placed at \$1800, the Jump Table is logically above A5. The program globals are located below A5. The details of the portion of memory addressed by offsets from A5 is:

```
| Jump Table
  n | Intrinsic Unit
                                           JT SEGMENT
   + Data Ptr
       Table
        of
     n Units
1 1 1
                       (distance to jump table)
  JTSegDelta
                      ----($13C)
  StkSegDelta
                       (distance to stack)
  | main program parameters| (see below)
   += = = = = = = = (A5)
  | Main program globals |
   STACK SEGMENT
  Regular Unit globals
+ - > | Intrinsic Unit Globals | (Shared Intrinsic Unit globals are elsewhere)
```

If the program is using shared intrinsic units, some of the intrinsic unit data pointers point to locations in the Shared Data Segment which contains global data used by all processes. The JT Segment is read-only and grows up. The stack segment is read-write and grows down.

The parameters to the main program are:

```
| pointer to $$FIRST | +58
                  +56
reserved
| Lisagraf info | +52
| Saved registers |
| Monitor flag | +21
| Physical Size | +20
| Common Size | +16 (regular and intrinsic unit size)
                  1 +12
1 @OUTPUT
| @INPUT
                  1 +8
| Return address | +4
| Old A5
                1 +0
                 ----<- - - - - (A5)
(Globals)
```

A6 is the stack frame pointer. The stack frame of a procedure is:

```
| Caller's stack frame |
| Caller's dynamic link | <--+ |
| Function Result (only |
| for a function) |
| Procedure arguments |
| Static Link (only for a |
| level 2 or higher proc) |
| Return Address |
| Local frame |
| Dynamic requirements |
| Low Memory | (A7)
```

Chapter 6 THE ASSEMBLER

5.1 The Assembler
6.2 Using the Assembler 6-3 The assembler accepts a text file as input, and produces a machine language (.OBJ) file as output.
6.3 Assembler Opcodes6-5 The assembler opcodes are the standard 68000 opcodes, with a few alternate forms for some instructions.
6.4 Assembler Syntax An assembler statement consists of an optional label, the opcode, and one or two operands. The operands can contain expressions.
6-5 Assembler Directives 6-10 The assembler directives provide for procedure and function definition, macros, label and constant declaration, listing control, storage allocation, and conditional assembly.
6-6 Communication with Pascal 6-17 Assembly language routines can be either procedures or functions called from a Pascal program. Parameters are passed on the Pascal stack.
6.7 Assembly Language Examples 6-21 This section provides example assembly language routines illustrating parameter passing and other functions.

The Assembler

Workshop User's Guide for the Lisa

THE ASSEMBLER

6.1 The Assembler

The assembler is a program that translates assembly language source code into object code. The assembler accepts a text file containing the source code as input, and produces an object file as output. The object file produced must be linked with a Pascal main program before it can be executed.

Assembly language routines are used to implement low level or time critical functions. This chapter describes how to use the assembler, and the syntax of assembly language programs. Information on the machine instructions available on the 68000 processor can be found in the Motorola MC68000 Reference Manual.

6.2 Using the Assembler

To assemble a program, press A from the Workshop command line. Then specify the input file (the file that contains your source program) and two output files: an optional listing file and the object file (the file that contains the object code produced by the assembler).

The input file must be a text file containing assembly language source statements. You can create this file with the Editor. The output file produced is an object file (.OBJ), that must be linked with a Pascal main program to be run.

6.2.1 Assembler Options

When you start the assembler, the option settings are displayed. You can enter the options selection mode by responding to the input file prompt with "?". There are two assembler options:

- P Pretty Listing.
- S Print information about available space.

Each option may be set to + or -:

- + On
- Off

When Pretty Listing is on, the forward referenced labels or offsets are filled in with the correct values

After setting options, press return, and the assembler asks you for the name of the input file. The assembler then asks you for the name of the listing, and the output files.

6.2.2 The Input File

The input file is a text file containing assembler language source statements. A file created using the Editor will be in text file format.

When the assembler asks you for the name of the input file, type "?" if you want to change assembler options at this time; otherwise type the pathname of your source file.

6.2.3 The Object File

The object file produced by the assembler contains a machine code version of your source program. The name of an object file ends with .OBJ. Araw assembly object file is not executable; it must be linked with a Pascal program that calls it. See Section 6.6 for further information.

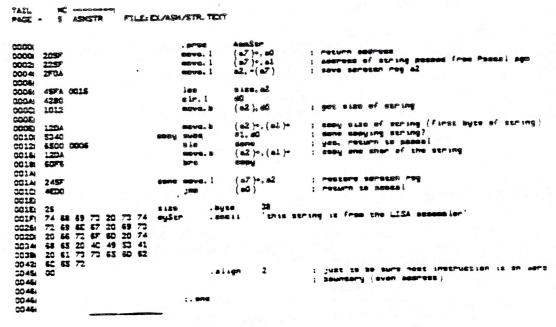
The output file will be an object file which must be linked with a Pascal main program before it can be executed.

6.2.4 The Listing File

The listing file produced by the assembler contains a list of source statements and their machine-language equivalent. If Pretty Listing is off, all addresses for forward referenced labels will be displayed as asterisks ("*). If Pretty Listing is on, the actual value will be filled in.

Source statement errors are flagged in the listing. Refer to the Appendix for a list of essembler error messages.

An example of an assembler listing file is shown in Figure 6-1. Figure 6-2 shows the same file listed with the Pretty list option.



Assembler Listing

If you specify a device name such as -PRINTER or -CONSOLE for the listing file, the listing will be printed on that device. If you specify a disk file, the listing will be created as a text file; you may then print it by using the Copy command in the File Manager command line.

NOTE

If you want pretty listing, the listing output must be sent to a file, not to a device.

pretty listing

Figure 6-2 Pretty Listing

6.3 Assembler Opcodes

The 68000 opcodes are described in the Motorola MC68000 Microprocessor User's Manual. The assembler has two variant mnemonics for branches (BHS for BCC and BLO for BCS). The variant names are more indicative of how the instruction is being used after unsigned comparisons. The default radix is decimal.

The size of an operation (byte, word, or long) is specified by appending either .B, .W, or .L to the instruction. The default operation size is word. To cause a short forward branch, append a .S to the instruction. The default branch size is Long.

Note that the TAS instruction does not work on the Lisa machine.

6.3.1 Optimization

Note that the Assembler accepts generic instructions and assembles the correct form. The instruction ADD, for example, is assembled into ADD, ADDA, ADDQ, or ADDI, depending on the context.

ADD D3, A5

becomes ADDA D3, A5.

MOYE, CMP, and SUB ere handled in a similar manner.

6.4 Assembler Syntax

This section describes the form in which the assembler expects an assembly language program. The structure of an assembly language program is described in Section 6.4.1. Rules for forming constants, identifiers, labels, expressions, and addressing modes are provided in the following sections.

6.4.1 Structure of an Assembly Language Program

An assembly language program contains one or more procedures or functions. The structure of an assembly language source file looks like Figure 6-3. The source file contains an (optional) section of operations that don't generate code. Constants or macros are usually defined here. Next it conains one or more procedures (.PROC) or functions (.FUNC). These each contain a sequence of code generating operations and directives. A procedure or function ends when the assembler encounters the next .PROC or .FUNC. A .END directive is the last statement in the program. Any text beyond the .END is ignored.

non code generating operations

.PROC (or .FUNC) code generating operations and any directives needed

.PROC

.FUNC

.END

Figure 6-3

Structure of an Assembly Language Program

The directives that don't generate code are:

LEQU	.MACRO	.IF .ELSE	LIST NOLIST	.MACROLIST .NOMACROLIST	
.REF	EF	ENDC	PAGE	.PATCHLIST	
.DEF			.TITLE	.NOPATCHLIST	

6.4.2 Constants

Constants in the assembler can be either numeric or string constants.

6.4.2.1 Numeric Constants

Numeric constants in the assembler can be expressed in decimal, hexadecimal, octal, or binary. The default radix is **decimal**. Numeric constants are expressed as follows:

Decimal

Decimal numbers are formed with the decimal digits (0-9). Examples:

10 13 137

Hexadecimal

Hex numbers can be expressed in two ways:

1. Preceed the number with a "\$". Examples of this are:

\$FF13 \$127

2. Follow the number with an "H". Using this form, the number must start with a digit (0-9). Examples:

0FF13H 195H

Octal

Octal numbers are followed by the character "O". Note that this is the letter O, not the number zero (0). Examples:

770 1040

Binary

Binary numbers are followed by the character "B". Examples:

1011B 111000B 6.4.2.2 String Constants

String constants are delimited by matching pairs of single or double quotes. Examples of string constants are:

"this is a string constant"

'using single quotes as delimiters lets you include "double" quotes'

6.4.3 Identifiers

Only the first eight characters of identifier names are meaningful to the assembler. The first character must be alphabetic; the rest can be alphanumeric, period, underbar, or percent sign.

Examples of identifiers are:

LOOP EXIT_PRC NUM num64%

6.4.4 Labels and Local Labels

Labels begin in column one. They can be followed by a colon, if you like.

Local labels can be used to avoid using up the storage space required by regular labels. The local label stack can handle 50 labels at a time. It is cleared every time a regular label is encountered. A local label is an a followed by a string of decimal digits (0-9). Examples of local labels are:

a123 a2 **a**79

6.4.5 Expressions and Operators

All quantities are 32 bits long unless constrained by the instruction. Expressions are evaluated from left to right with no operator precedence. Angle brackets can be used to control expression evaluation. The operators are:

- unary or binary addition
- unary minus or subtraction
- ones complement (unary operator)
 - exclusive or
- multiplication
- division (DIV)
- MOD
- logical OR
- logical AND 8
- equal (used only with .IF)
- not equal (used only with .IF) <>

There is no operator precedence in expressions. For example, in the expression 2+9*4, the addition is performed first. To perform the multiplication first, rewrite the expression with angle brackets to show precedence: $2+\langle 9*4 \rangle$; or reorder the operands: 9*4+2.

6.4.6 Addressing Modes

Refer to the Motorola 68000 manual for detailed information on the addressing modes supported by the 68000 microprocessor. Table 6-1 gives a summary of the addressing modes including their syntax.

Table 6-1. Summary of Addressing Modes

Mode	Register	Syntax	Meaning E	xtra Word	s
0	07	Di	Data direct	0	
1	07	Ai	Address direct	0	
2	07	(Ai)	Indirect	0	
3	07	(Ai)+	Postincrement	0	
4	07	-(Ai)	Predecrement	0	
5	07	e(Ai)	Indexed	1	
6	07	e(Ai,Ri)	Offset indexed	1	
7	0	e	Absolute short addre	ss 1	
7	1	e	Absolute long addres		
7	2	е	PC Relative	1	
7	3	e(Ri)	PC Relative indexed	1	
7	4	≇e	Immediate	1 or 2	

Notes:

The indexed and PC relative indexed modes are determined by the opcode.

The absolute address and PC relative address modes are determined by the type of the label (absolute or relative).

The absolute short and long address modes are determined by the size of the operand. Long mode is used only for long constants.

The number of extra words for immediate mode is determined by the opcode size modifier (.W or .L).

6.4.7 Miscellaneous Syntax

Comments

A comment in an assembly language program begins with a semicolon. All characters on a line after a semicolon are ignored. Examples are:

; This is a comment on a line by itself CLR.L DO ; comment after a statement

Current Program Location

The current program location is indicated in assembly language by the symbol "*". Examples of its use are:

Loop infinitely Jump back 4 bytes JMP *-4

Move Multiple (MOYEM)

To specify which registers are affected by Move Multiple (MOVEM), specify ranges of registers with "-", and specify separate registers with "/". For example, to push registers DO through D2, D4, and A0 through A4 onto the top of the stack:

MOVEM . L DO-D2/D4/A0-A4, -(A7)

6.5 Assembler Directives

Assembler directives tell the assembler to do various functions besides providing directly executable code. These functions include defining symbols and constants, defining macros, doing conditional assembly, and controlling listing options.

The Assembler directives (pseudo-ops) are shown in Table 6-2.

Table 6-2 The Assembler Directives

.FU	ROC INC IF IF IF	Operands <identifier> <identifier> <identifier-list> <identifier-list> <identifier-list> '<name>'</name></identifier-list></identifier-list></identifier-list></identifier></identifier>	Meaning begin procedure begin function make identifiers externally available declare external identifiers put following code in segment 'name' end of entire assembly
.BY .BL .W	CII CTE OCK ORD ONG LIGN	<pre>'<character-string>' <value-list> <length>[value] <value-list> <value-list> <expr></expr></value-list></value-list></length></value-list></character-string></pre>	place ASCII string in code allocate a byte in code for each value allocate length bytes of value allocate a word for each value allocate a long word for each value allign next code on multiple of Expr
	RG ORG	<value></value>	place next byte at (value) same as .ORG
.EC	Ų	<value></value>	set label equal to <value></value>
	ACRO NDM	<identifier></identifier>	begin macro definition end macro definition

Table 6-2 (continued)

The Assembler Directives

Directive Operands .IF 〈expr〉 .ELSE .ENDC	Meaning begin conditional assembly optional alternate to .IF block end conditional assembly
.LIST .NOLIST .PAGE .TITLE ' <title>' .MACROLIST .NOMACROLIST .PATCHLIST .NOPATCHLIST</td><td>turn on assembly listing turn off assembly listing issue a page feed in listing title of each page in listing turn on macro expansion listing turn off macro expansion listing turn on patchlist turn off patchlist</td></tr><tr><td>.INCLUDE (filename)</td><td>insert (filename) into assembly</td></tr></tbody></table></title>	

6.5.2 Space Allocation Directives

The space allocation directives are .ASCII, .BYTE, .WORD, .LONG, and .BLOCK.

.ASCII 'string'

Converts 'string' into the equivalent ASCII byte constants and places the bytes in the code stream. The string delimiters must be matching single or double quotes. To insert a single quote into the code use double quotes as delimiters. Similarly for double quotes:

```
.ASCII "don't" ; string containing single quote .ASCII 'a "glitch" ; string containing double quote
```

BYTE (values)

Allocates a byte of space in the code stream for each of the values given. Each value must be between -128 and 255.

BLOCK (length)[value]

allocates (length) bytes, each filled with the value given. If no value is given, a block of zeroes is allocated.

WORD (values)

allocates a word of space in the code stream for each of the values listed. The values must be between -32768 and 65535.

For example.

TEMP . WORD

0, 65535, -2, 17

creates the assembled output:

00000

FFFF

FFFE

0011

LONG (values)

Allocates two words of space for each value in the list. For example,

STUFF . LONG 0, 65535, -2, 17

creates the output:

000000000

0000FFFF

FFFFFFE

00000011

(label) EQU (value)

Assigns (value) to (label). (value) can be an expression containing other labels.

.ORG (value)

Puts the next byte of code at <value> relative to the beginning of the assembly file. Bytes of zero are inserted from the current location to (value).

RORG

is similar to .ORG. It indicates that the code is relocatable. Because the loader does not support absolute loading, .ORG and .RORG accomplish the same function. All addressing must be PC relative.

RORG (without the leading period) is the same as .RORG. Similarly, END = .END, EQU = .EQU, PAGE = .PAGE, LIST = .LIST, NOL = .NOLIST.

6.5.3 Macro Directives

A macro consists of a macro name, optional arguments, and a macro body. When the assembler encounters the macro name, it substitutes the macro body for the macro name in the assembly text. Wherever %n occurs in the macro body (where n is a single decimal digit), the text of the n-th parameter is substituted. If parameters are omitted, a null string is used in the macro expansion. A macro can invoke other macros up to five levels deep. In the assembly listing, the listing of the expanded macro code is

controlled by the options .MACROLIST and .NOMACROLIST. These options are described in Section 6.5.6.

```
.MACRO <identifier>
.
.ENDM
```

defines the macro named (identifier). The following is an example of a macro:

```
.MACRO Help
MOVE %1,DO
ADO DO,%2
.ENDM
```

If "Help" is called in an assembly with the parameters "Alpha" and "Beta", the listing created would be:

```
# MCVE Alpha, Beta
# MCVE Alpha, DO
# ADO DO, Beta
```

6.5.4 Conditional Assembly Directives

The conditional assembly directives .IF, .ELSE, and .ENDC are used to include or exclude sections of code at assembly time based on the value of some expression.

JF (expression)

identifies the beginning of a block of source code that is assembled only under certain conditions. If <expression> is false, the Assembler ignores code until a .ELSE or .ENDC is found. The code between the optional .ELSE and .ENDC is assembled if <expression> is false. Otherwise it is ignored.

<expression> is considered to be false if it evaluates to zero. Any non-zero
value is considered true. The expression can also involve a test for equality
(using <> or =). Strings and arithmetic expressions can be compared.
Conditionals can be nested. The macros HEAD and TAIL given in section
6.6.1 provide examples of the use of conditionals. The general form is:

6.5.5 External Reference Directives

Separate routines can share data structures and subroutines by linkage between assembly routines using .DEF and .REF. These directives generate link information that allows separately assembled routines to be linked together.

.DEF and .REF directives associate labels between assembly routines, not between assembly routines and Pascal. The only way to communicate data between Pascal and assembly routines is by using the stack. This is done by passing the data as parameters in the procedure or function call. Information on parameter passing between Pascal and assembly language is found in section 6.6.

DEF (identifier-list)

identifies labels defined in the current routine as available to other assembly routines through matching .REFs. The .PROC and .FUNC directives also generate code similar to that generated by a DEF with the same name, so assembly routines can call external .PROCs and .FUNCs with REFs.

> .PRCC Simple, 1 Alpha, Beta DEF

Reta BNE

Alpha MOVE

RTS

Beta MOYE

RTS

This example defines two labels, Alpha and Beta, which another assembly routine can access with .REF.

REF (identifier-list)

identifies the labels in (identifier-list) used in the current routine as available from some other assembly routines which defined these identifiers using the .DEF directive.

.PROC Simple .REF Alpha

JSR Alpha

END

uses the label 'Alpha' declared in the .DEF example.

When a .REF is encountered, the assembler generates a short absolute addressing mode for the instruction (the opcode followed by a word of 0's) and a short external reference with an address pointer to the word of 0's following the opcode. If the referenced label and the reference are in the same segment module, the Linker changes the addressing mode from short absolute to single-word PC relative. If, however, the referenced procedure is in a different segment, the Linker converts the reference to an indexed addressing mode (off A5) and the word of zeros is converted into the proper entry offset in the jump table. If the referenced procedure is in an intrinsic unit (and therefore in a different segment), the IUJSR, IULEA, IUJMP, and IUPEA instructions are used. The Linker blindly assumes that the word immediately before the word of zeros is an opcode in which the low order 6 bits are the effective address. Thus, a .REF label cannot be used with any arbitrary instruction. The .REF labels are intended for JSR, JMP, PEA, and LEA instructions.

SEG

default segment name is " (8 blanks). .SEG "segment name" puts the code in segment called "segment name".

6.5.6 Listing Control Directives

The directives that control the Assembler's listing file output are .LIST, .NOLIST, .PAGE, .TITLE, .MACROLIST, .NOMACROLIST, .PATCHLIST, and .NOPATCHLIST. If you do not specify a name for the listing file in response to the assembler's prompt the listing directives are ignored.

The default for the assembler is for .LIST, .MACROLIST, and .PATCHLIST to be in effect when the assembler starts. .TITLE defaults to blank.

LIST and NOLIST

can be used to select portions of the source to be listed. The listing goes to the specified output file when .LIST is encountered. .NOLIST turns off the listing. .LIST and .NOLIST can occur any number of times during an assembly.

PAGE

causes the next line of the listing file to be printed on the next page.

.TITLE '<title>'

specifies a title for the listing page. <title> can contain up to 80 characters, and can be enclosed in either single or double quotes. For example:

.TITLE 'Interpreter'

places the word, interpreter, at the head of each page of the listing.

.PATCHLIST

patches the forward referenced labels in the listing. It must be on if you want Pretty Listing.

NOPATCHLIST

turns off patching of forward references.

MACROLIST

turns on listing of the expanded code from a macro.

.NOMACROLIST

turns off listing of macro expansion. See Figure 6-4 for examples of the macro listing options.

macro listing

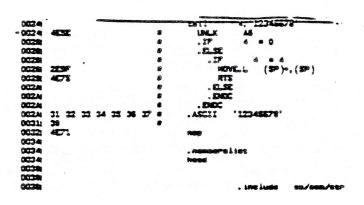


Figure 6-4
Mecro Listing Options

6.5.7 File Directives

The pseudo-op

INCLUDE (filename)

causes the contents of (filename) to be assembled at the point of the INCLUDE. You need not specify the .TEXT suffix. An included file cannot itself contain an .INCLUDE statement.

6.6 Communication with Pascal

Assembly language routines must be called from a Pascal program. In order to call an assembly language routine, the Pascal program declares the assembly language procedure or function to be EXTERNAL. If the assembly routine does not return a value, use .PROC. If .FUNC is used, space for the returned value is inserted on the stack just before the function parameters, if any. The amount of space inserted depends on the type of the function. A Longint or Real function result takes two words, a Boolean result takes one word with the result in the high order byte, and other types take one word.

In the following example, an assembly language routine is linked to a Pascal program. The assembly language routine accepts two integers and returns the logical AND of them. The Pascal host file is:

```
PROGRAM BITTEST:
YAR I, J: INTEGER;
FUNCTION I and (i, j: INTEGER): INTEGER;
                   (* external = Assembly language *)
  EXTERNAL:
BEGIN
  i := 255;
  j := 33;
  WRITELN (I, J, 'AND = ', I and (I, J));
END.
The Assembler file is:
      .FUNC
                IAND
      MOYE.L
                (A7)+, A0
                               ; return address
      MOVE . W
                               ; J
                (A7)+, D0
      MOVE .W
                (A7)+,D1
                D1, DO
                               ; I AND J
      M. DIA
               DO, (A7)
      MOVE . W
                               ; put function result on stack
      MP
                (AO)
      .END
```

In the example given above little attempt has been made to make the assembly language procedure mimic the structure of a procedure generated by the Pascal Compiler. A complete description of this structure requires some preliminary discourse.

6.6.1 The Run Time Stack

Automatic stack expansion code makes procedure entries a little complicated. To ensure that the stack segment is large enough before the procedure is entered, the compiler emits code to 'touch' the lowest point that will be needed by the procedure. If we 'touch' an illegal location (outside the current stack bounds), the MMU hardware signals a bus error that causes the 68000 to generate a hardware exception and pass control to an exception handler. This code, provided by the operating system, must be able to restore the state of the world at the time of the exception, and then allocate enough extra memory to the stack that the original instruction can be re-executed without problem. To be able to back up, the instruction that caused the exception must not change the registers, so a TST.W instruction with indirect addressing is used.

In the normal case, the procedure's LINK instruction should be preceded by a TST.W e(A7) which attempts to reach the stack location that can accompose the static and dynamic stack requirements of the procedure. If the static and dynamic stack requirements of your assembly language procedure are less than 256 bytes, you can assume that the compiler's fudge factor will protect the assembly language procedure, so the TST.W can be omitted. If the requirements are greater than 32K bytes, e(A7) may not be sufficient because only 16 bits of addressability are available. In this case, the compiler currently emits code something like:

MOVE.L A7, A0
SUB.L #Size, A0 ; #size=dynamic + static needed
TST.W (A0)

If the compiler option D+ is in effect (the default), the first eight bytes of the data area following the final RTS or JMP (A0) contain the procedure name. The debugger gets the procedure name from this block, allowing you to use procedure names in the Debugger. The following example is provided to show how an assembly language programmer can provide the Debugger with all the information it needs to perform fully symbolic low level debugging. Note that all procedure names must be in upper case to be compatible with the Debugger.

```
; ASSEMBLY LANGUAGE EXAMPLE
 DEBUGF .EQU 1
                                ; true => allow debugging with
                                ; proc names
    HEAD — This MACRO can be used to signal the
    beginning of an assembly language procedure. HEAD
    should be used when you do not want to build a stack
    frame based on A6, but do want debugging information.
    No arguments
      .MACRO HEAD
        .IF
               DEBUGF
            LINK
                     A6, #0
                                  ; fancy NOP used by debugger
        .ENDC
       .ENDM
    TAIL — This MACRO can be used as a generalized exit
    sequence. There are two cases. First, if you build a stack frame, TAIL can be used to undo the stack
    frame, delete the parameters (if any) and return.
    Second, if you do not want to build a stack frame
    based on A6, this MACRO can be used to signal the
    end of an assembly language procedure. In either
    case if DEBUGF is true the Procedure_name is dropped by the MACRO as an 8-character name.
    Two arguments:
            1) Number of bytes of parameters to delete
            2) Procedure Name as string exactly 8 characters
       .MACRO TAIL
         UNLK
                   A6
          .IF
                   %1 = 0
            RTS
                                     ; O bytes of parameters
          .ELSE
                      %1 = 4
               MOVE.L (A7)+, (A7); 4 bytes of parameters
                RTS
             .ELSE
               MOVE . L
                         (A7)+, AO ; put return addr into AO
                         #%1, A7 ; remove params from stack
                ADD.W
```

```
; return to caller
                   (AO)
          MP
        .ENDC
     .ENDC
             DEBUGF
     .IF
        . ASCII
     .ENDC
  .ENDM
The following example demonstrates the use of the
TAIL macro for the purpose of debugging. The example
assumes that you want to build a stack frame based
on A6. In a real assembly language procedure the
zeroes below would be replaced by the local size and
peremeter size.
  .PROC
           SIMPLE
                         ; zero bytes of locals
  LINK
           A6, #0
                         ; body of procedure
  NOP
                         ; zero bytes of parameters
  TAIL
  ONG.
```

These macros are sufficient totall small assembly language routines from Pescal.

Upon entry to the assembly routine, the stack is as shown in Figure 6-5.

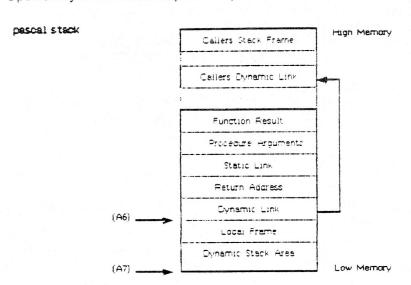


Figure 6-5
The Pascal Run Time Stack

The function result is present only if the Pascal declaration is for a function. It is either one or two words. If the result fits in a single byte (a boolean, for example), the most significant half (the lower addressed half) gets the result value.

Parameters are present only if parameters are passed from Pascal. They are pushed on the stack in the order of declaration. All reference parameters are represented as 32 bit addresses. Value parameters less than 16 bits long always occupy a full word. All non-set value parameters larger than 4 bytes are passed by reference. It is the procedure's responsibility to copy them. All large set value parameters are pushed onto the stack by the calling routine.

The static link is present only if the external procedure's level of declaration is not global. The link is a 4 byte pointer to the enclosing static

It is the responsibility of the assembly language procedure to deallocate the return address, the static link (if any), and the parameters (if any). The SP must point to the function result or to the previous top of the stack upon return. Registers D4 through D7 and A3 through A7 must be preserved. We recommend that you also preserve D3 and A2.

6.6.2 Register Conventions

The following are the register conventions used in the Lisa system. It is your responsibility to preserve these registers.

DO-D2/AO-A1: Scratch registers (can be clobbered) D3, A2: Scratch registers, but should be preserved D4-D7/A3, A4: Used for code optimization (must be preserved) A5: Pointer to user globals (must be preserved) A6: Pointer to base of stack (must be preserved)

Top of stack SP:

Registers D3 and A2 may be used at some time in the future by the compiler for code optimization, so you should preserve them also.

6.7 Assembly Language Examples

The following examples show how to use certain features of the assembly language.

```
6.7.1 Using .REF and .DEF Directives
    The first example illustrates the use of .REF and .DEF. These two
    directives allow an assembly language routine to reference another
    assembly routine.
    The Pascal host file is:
    program WasteTime;
    procedure Wait (time : integer);
        external;
    begin
        writeln ('Going to waste some time');
        wait (50);
        writeln ('Finished wasting time');
    end.
    The assembly language file is:
                   wait
           .proc
                                  ; need to use a piece of code
          .ref
                   cycle
                                  ; whose entry point is cycle
                                  ; defined outside procedure wait
                                  ; another outside procedure
           .ref
                   more_time
                                  ; return address in a0
                    (87)+,80
          move.l
                                  ; need to wait this many cycles
                   (87) + 00
          w. svom
                                  ; a parameter for cycle
                    cycle
           jst
                                  ; waste more time
           jst
                    more_time
                                   ; return
           gmt
                    (a0)
          ; the subroutine used by wait is defined in the
           ; following code. this proc could do other things
           ; besides the cycle routine
                    def_cycle
           .proc
                                   ; cycle visible to other procs
           . def
                    cycle
              code can go here
                                   ; example of a line of code
           nop
                                   ; beginning of the cycle routine
     cycle
                                   ; parameter is in dO
                    #1, d0
           sub
           bne
                    cycle
           rts
           ; more code can go here
```

```
.proc more_time ; waste more time clr dO ; use dO as timer @1 add #2, dO bne @1 rts .end
```

6.7.2 String Parameters

The following program illustrates how to pass a Pascal string to an assembly language program, modify the string, and return it. Pascal strings have their length stored as the first byte in the string.

The Pascal source file is:

```
program pasStr;
        strType = string[80];
type
var
        str : strType;
        ch : char;
procedure AsmStr (var str : strType);
  external;
begin
  str := 'initial string in Pascal main program';
  writeln (str);
  AsmStr (str);
  writeln (str);
  writeln;
  write ('press any key to continue');
  read (ch);
```

The assembly language file is:

```
AsmStr
      .proc
     move.1
               (A7)+, A0
                              ; return address saved in AO
                              ; address of string from Pascal
               (A7)+, A1
     move.1
              A2, -(A7)
                              ; save scratch register A2
     move.1
               size, A2
     lea
     clr.l
               \infty
     move.b
             (A2), DO
                              ; get size of string
               (A2)+, (A1)+
                              ; copy size of string
     move.b
                              ; done copying string?
               #1.DO
copy subq
     blo
               done
                              ; yes, return to Pascal
```

```
move.b (A2)+, (A1)+ ; one cher of string
          bra
                   copy
                                  restore scratch register
                   (A7)+,A2
    done move.l
                                  ; return to Pascal
                   (AO)
          amr.
    size .byte
                   38
                   'this string is from the LISA assembler'
    myStr .ascii
                                ; get on a word boundary
          .align
6.7.3 Writing a Function
    The following example shows how to write a function in assembly language.
    This function returns a boolean value.
    The Pascal program is:
    program booleanFunction;
           int : integer;
    Var
           ch : char;
    function swapBytes (var int : integer) : boolean;
      external:
    begin
      int := 256:
      writeln ('the initial value of int = ', int:1);
      repeat
        if swapBytes(int) then
          writeln ('int = ', int:1)
       else writeln ('int = 0, function value is false');
        int := int - 1;
      until (int < 0);
      write ('press any key to continue');
      read (ch);
    end.
    The Assembly language function is:
          .func
                   swapBytes
          move.1
                   (A7)+A0
                                  ; pop return address
                                  ; get address of word to swap
                   (A7)+A1
          move.l
                                  ; get the number
                   (A1), DO
          move
                                  ; swap the bytes
          ror
                   #8,DO
          move
                   DO, (A1)
                                  ; put it back
                   91
          bne
                                 ; number = 0 so return false (0)
                   (A7)
          clr
```

```
bra
                   2
    @1
                    #$FFFF, (A7)
          move
                                   ; return result true (non zero)
    22
          jmp
                    (AO)
                                   ; return to calling program
          .end
6.7.4 Calling Pascal I/O Routines
    The following example illustrates how to call Pascal routines from
    assembly language to do I/O. Note the use of macros for calling the Pascal
    routines.
    program APLpascal;
    type
           strType = string[80];
    var
            str:strType;
            f1, f2: text;
            ch: char;
    procedure main;
      external;
    function f_rewrite (f_num: integer; f_name: strType):integer;
    begin
      case f_num of
        1: rewrite (f1, f_name);
        2: rewrite (f2, f_name);
      f_rewrite := ioresult;
    end;
    function f_reset (f_num: integer; f_name: strType): integer;
    begin
      case f_num of
        1: reset (f1, f_name);
        2: reset (f2, f_name);
      end;
      f_reset := ioresult;
    end;
    procedure writeLine (f_num: integer; var S: strType);
    begin
      case f_num of
        0: write (s);
1: write (f1, s);
                              {file id = 0 means write to -console}
         2: write (f2, s);
```

```
end;
end;
procedure writeLF (f_num: integer; var S: strType);
begin
 case f_num of
   O: writeln (s);
    1: writeln (f1, s);
    2: writeln (f2, s);
  end;
end;
procedure f_close (f_num: integer; lock_file: boolean);
begin
 case f_num of
1: if lock_file then
         close (f1, lock)
       else
         close(f1);
    2: if lock_file then close(f2,lock)
       else close(f2);
  end:
end;
  writeln ('test program - using assembly main routine to do I/O');
 writeln;
 main;
 write ('press any key to continue'); {so user can know it is done}
  read (input, ch);
end.
             .proc
                       main
             .ref
                       writeLF
             .ref
                       writeLine
                       frewrite
             .ref
             .ref
                       f_reset
                       f_close
             .ref
                                      ; id # of file one
             .equ
first_file
                       2
                                      ; id # of file '-printer'
printerId
             .equ
```

```
; return address to the Pascal main routine is left on the stack
             .macro
                     open_write_file
                 %1 — file # %2 — file name
                                     ; reserve space for function
             clr
                      -(a7)
                                     ; result from f_rewrite
; file id # as first param
             move
                      \#%1, -(a7)
                                     ; second param is file name
             lea
                      %2, a0
                      a0, -(a7)
             move.l
                      f_rewrite
             jsr
                      (a7)+,a0
                                     ; pop IOresult
             move
             ble
                      31
                      %2
                                     ; IOresult > 0 -> error
             error
                                     ; (nested macro call)
81
             .endm
             .macro open_read_file
                  %1 — file #
                  %2 --- file name
                                     ; reserve space for function
             clr
                      -(a7)
                                     ; result of f_reset
             move
                      #%1, -(a7)
             lea
                      %2, a0
                      a0, -(a7)
             move.1
             jsr
                      f reset
                                     ; pop IOresult
                      (a7)+, a0
             move
             ble
                      91
                      %2
             error
                                     ; IOresult > 0 -> error
             endm
01
             .macro
                      write_file
                                    ; write a line (with no linefeed)
                  %1 --- file #
                  %2 — label of string to be written
                       #%1, -(a7)
             move
             lea
                      %2, a1
                                     ; push string address onto stack
             move.1
                       a1,-(a7)
             jsr
                      writeLine
                                     ; write it out
             .endm
             .macro writeLn_file ; write a line of text with linefeed
                  %1 --- file #
```

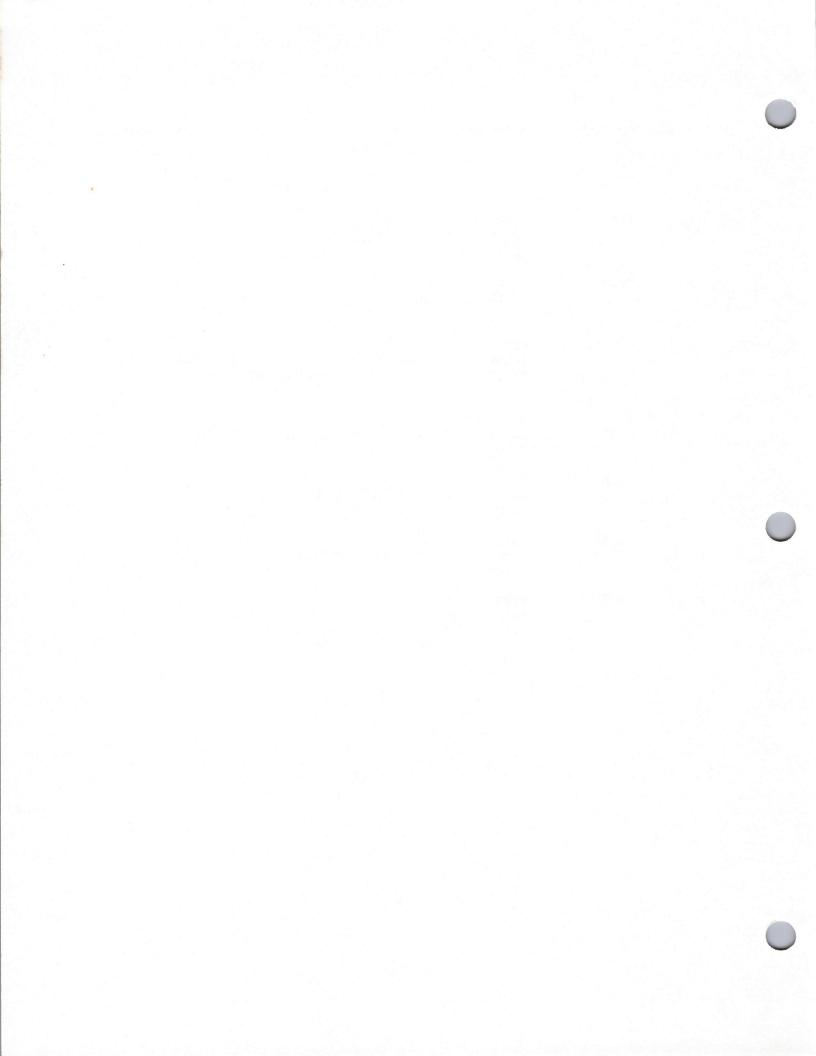
%2 --- label of string to be written

```
#%1, -(87)
           move
                  %2, 31
           lea
                               ; push string address onto stack
                   a1, -(a7)
           move.1
                               ; write it out
                   writeLF
           jst
           .endm
           .macro close_file
              %1 -- file #
               %2 -- close status code
                                      normal close
                     0 - $00ff
                     $0100 - $ffff
                                      lock
                  #%1, -(a7)
           move
                  #%2, -(87)
           move
                   f_close
           jsī
           .endm
           .macro error
           ; %1 — file name
                              ; write error message
           write_file 0, errStr
                               ; to -console
                               ; (file id # 0)
                               ; output file name also
           writeLn_file 0, %1
           rts
                                ; quit
           .endm
MAIN ASSEMBLY LANGUAGE PROGRAM
open_write_file first_file, file1
                                      ; open IO/read.text
      open_write_file printerId, printer
                                      ; write the openstr
     writeLn_file 0, openstr
                                      ; to -console (file # 0)
                    first_file, string ; write string to first_file
      writeLn file
                                      ; write str1 to printer
     writeLn_file
                    printerId, str1
                     first_file, $0100 ; lock first_file
      close file
                     printerId, 0
                                    ; do not lock the printer
      close_file
                                     ; no error should occur
      open_read_file 1, file1
                     1, $ffff
                                     ; preserve file1
      close_file
                                     ; no errFile around, should
      open_read_file 2, errFile
                                      ; cause error.
```

rts

; back to pascal main program

file1 .byte 14 'ID/record.text' .ascii .align printer .byte '-printer' .ascii .align string .byte 'this string is from the LISA assembler' .ascii .align ; make sure on even memory stri .byte myStr .ascii 'another string from LISA assembler' .align openstr .byte 'opened file IO/record.text' .ascii .align errStr .byte 'error in opening file ' .ascii .align errFile .byte 6 'noFile' .ascii .align 2 .end



LISABUG

LisaBug allows you to examine, disassemble, and change the contents of memory, set breakpoints, and do immediate assemblies. If the compiler D option is on (the default), procedure names are available to the debugger, and Lisabug uses the symbols wherever appropriate.

Type M to the Monitor command prompt to invoke LisaBug. It asks:

What file?

You can type <cr> to enter LisaBug without any file. If you type a file name, that code file is loaded into LisaBug. The LisaBug command prompt is '>'. The default radix is hexadecimal.

You can drop into LisaBug by hitting the NMI key which is currently the third key from the left in the top row of the numeric keypad.

A FEW EXAMPLES

If you type a file name, LisaBug starts up with the program counter at the start of the program. To see one instruction disassembled (say at 32F96), type

>ID 32F96

(followed by RETURN, of course). ID stands for Immediate Disassemble. Each subsequent ID command, if given without any address, disassembles the next instruction found. In addition to printing the value of each byte, LisaBug prints the ASCII equivalent of that value, if a printable one exists. If none exists, it prints a period.

To disassemble 20 consecutive addresses, type

>IL

IL (Immediate Disassemble Lines) can also be followed by an address. Subsequent IL commands disassemble successive blocks of 20 consecutive locations in memory.

If the object file being examined was compiled with the D+ compiler option, the procedure names are available in LisaBug and can be used in any expressions. For example,

>IL Foo 5

disassembles the first 5 lines of procedure 'Foo'.

>BR Foo+40

sets a break point 40 bytes into procedure 'Foo'.

You can also use labels in immediate assemblies:

>sy Ken 6000 >A Ken NOP

assembles a NOP instruction at the address 'Ken'.

>A 6000 <cr>> >Rich: TAS \$100 > <cr>

enters the immediate assembler at 6000, defines the label 'Rich', and assembles a TAS instruction.

THE SYMBOL TABLE

The symbol table is the union of the user symbol table and the distributed procedure names. The user symbol table contains the user declared symbols (like 'KEN' in the example above) and the predefined symbols (RDO and friends). Each entry contains twelve bytes. The first eight bytes are the symbol name, and the last four bytes are the symbol's value. Location \$406 gives the beginning of the symbol table, and \$40A points to the end of the table. The section 'Communication with Pascal' in the Assembler chapter of this manual contains more information about the symbol table.

LISABUG COMMANDS

Definitions:

Constant A constant in the default base

\$Constant A hex constant

&Constant A decimal constant

'ASCII String' An ASCII string

Name A symbol in the symbol table

RegName RDO..RD7, RAO..RA7, PC, US, or SS.

A predefined symbol in the symbol table with a value set by LisaBug. The value is equal to the value of the

register in question. LisaBug

automatically updates the values of these symbols. The 'R' is appended to distinguish the register names from hexadecimal numbers.

Expr An expression. Expressions can contain

names, regnames, strings, and constants.
Legal operators are + - * /. Expressions
are evaluated left to right. * and /
take precedence over + and -. (and)
can be used to indicate indirection.
< and > can be used to nest expressions.
In those cases where an odd value is probably

In those cases where an odd value is probab a mistake, LisaBug warns you that you are trying to use an odd address. If you decide to go ahead, it subtracts one from the address given. If the compiler option D+ is used, procedure names are also legal

in expressions.

Exprlist A list of expressions separated by blanks.

Register DO..D7, AO..A7, PC, SR, US, or SS. Note

that A7 is SP (the stack pointer).

Moving the LisaBug Window:

P expr Set port number to expr. Valid port numbers are:

O Lisa keyboard and screen (default)

1 UART Port A (farthest from Power Supply)

2 UART Port B

If you move the port to a UART, you must have a modem eliminator connected to that port.

Symbols and Base Conversion:

SY Display the values of all symbols

SY name Display the value of the symbol name

SY name expr Assign expr to the symbol name

CV exprlist Display the value of each expression in hex and decimal.

SH Set the default radix to hex

SD Set the default radix to decimal

Assembly and Disassembly:

A expr statement

A expr

Assemble one statement (instruction) at expr. If you use the form A expr, LisaBug asks you for the statement to be assembled. You can continue assembling instructions into consecutive locations.

Type (cr) to exit the immediate assembler.

ID Disassemble one line at the next address

ID expr Disassemble one line at expr

IL Disassemble 20 lines at the next address

IL expr Disassemble 20 lines starting at expr

IL exprl expr2 Disassemble expr2 lines starting at expr1

Upon entering LisaBug, the 'next address' is the current PC.

Set, Display, and Find Memory:

SM exprl exprlist

Set memory with exprlist starting at exprl. SM assumes that each element of exprlist is 32 bits long. To load different length quantities, use SB or SW described below. If the expression given is longer than 32 bits, SM takes just the upper 32. For example, if we ask LisaBug to:

SM 1000 'ABCDE'

it deposits the ASCII equivalent of 'ABCD' starting at 1000.

SB exprl exprlist

Set memory in bytes with exprlist starting at exprl

SW exprl exprlist

Set memory in words with exprlist starting at exprl

SL exprl exprlist

Set memory in long words with exprlist starting at exprl. For example,

SL 100 1

is equivalent to

SM 100 0000 0001

Display memory at expr. DM RA3+10, for example, displays the contents of memory from the address pointed to by A3 for 10 bytes. DM (110) displays the contents of the memory location addressed by the contents of location 110.

DM exprl expr2 Display memory. If exprl < expr2, then display memory from exprl to expr2. Otherwise, display memory for expr2 bytes starting at exprl.

DB expr Display memory as bytes.

DW expr Display memory as words.

DL expr Display memory as long words.

FB starting_addr count data Find Byte.

Find the byte or bytes 'data' in memory between 'starting_addr' and 'starting_Addr'+'count'.

FM starting_addr count data Find Memory

FW starting_addr count data Find Word

FL starting addr count data

Find Long

Set and Display Registers:

TD Display the Trace Display at the current PC

register

Display the current value of the register.

DO, for example, is a command to LisaBug to display the current value in the register DO.

RDO, on the other hand, is a name automatically placed in the symbol table to give you a handle

on the contents of DO in an expression.

register expr Set the register to expr

Memory Management:

LP expr Convert logical address to physical address.

DO expr Set the SEGI/SEG2 bits. These bits determine the hardware domain number. If the Status Register shows that you are in supervisor state, then the effective domain is zero, and the domain number

returned by LisaBug is the domain which would be active if the SR were changed to user state.

WP 0 or 1 Diable (0) or Enable (1) Write Protection. The default is 1.

MM start [end or count]

MM with one or two arguments displays information about the MMU registers. The second argument defaults to 1. If the starting address is greater than the second argument, the second argument is a count of the number of MMU registers to be displayed. If the starting address is less than the second argument, the second argument is the last register displayed.

MM 70

displays

Segment[70] Origin[000] Limit[00] Control[C]

These values are the Segment Origin, Limit, and Control bits stored by the hardware for each MMU register. As can be seen from a careful perusal of the hardware documentation, a Control value of C means the segment in question is unused (invalid). If the Control value is valid (F, for example), the debugger also displays the Physical Start

and Stop addresses of the segment.

MM &100 8

displays the MMU register information for the 8 registers starting at register 64 (decimal 100).

MM num org lim cntrl

The MM command followed by four arguments sets the MMU information for segment 'num'. The Origin, Limit, and control bits can be changed. The Monitor uses the first 16 registers, so it is safer not to mess with them.

MM 70 100 ff 7

sets the Origin of segment 70 to 100 and the control bits to 7 (a regular segment). The segment limit of -1 makes the segment 512 bytes long.

Breakpoints, Patchpoints, Traces, Calls:

BR

Display the breakpoints currently set. Up to 16 breakpoints can be handled by LisaBug. Break points are displayed both as addresses and as symbols. An asterisk marks the point of the breakpoint in the disassembly. Patch points are marked with '!'.

BR exprlist

Set each breakpoint in exprlist. Symbols are legal, of course, so we can:

BR Ralph+4

if Ralph is a known symbol.

PA insertion_addr destination addr

Insert a Patch. PA can be used to insert a sequence of code terminated by a TRAP #Sf into another sequence of code. Lisabug maintains a table of patches and return addresses to implement this facility. The trace command works with patches. It displays the next instruction to be executed and its environment. You can have up to 16 patches. A patch can be removed by using the CL (Clear) command with the patch insertion address.

PA Display patch addresses

CL Clear all breakpoints and patchpoints

CL exprlist Clear each breakpoint or patchpoint in exprlist

G	Start	running	at	the	current	PC

G expr Starting running at expr

Trace one instruction at the current PC

T expr Trace one instruction at expr

CA expr Call a subroutine in the debugger's environment.

SC expr Stack Crawl. Display the user call chain. Expr sets the depth of the display. It can be omitted.

Q Exit LisaBug, if it was called from Talk

RM Return to the Monitor. RM checks the interrupt level, stops exec files, and sets the domain to zero. If you are in an interrupt handler, RM may refuse to do anything. If the SR shows 2mxx where n is not zero, you are in an interrupt handler. To get back to the monitor, type G, hit NMI, and try RM again. With any luck, you

will escape eventually.

RB Reboot. The Lisa is reset. Reboot the Apple II and the Lisa should also reboot automatically.

Overcome Inadequate Hardware:

DU expr Disk unclamp. The Twiggy may not reliably eject the disk at the right time. If you have trouble, try the DU command followed by the drive number. Valid drive numbers are 1 and 2.

DC expr Disk Clamp. If the Twiggy refuses to suck up the disk and clamp it in place, try the DC command followed by the drive number (1 or 2).

RS Display the patch Return address Stack

If you have the debug card,

DR Display index or ranges of dump RAM.

MR Set a value level #5 interrupt on a word change.

```
register
                    Display the current value of the register.
register expr
                    Set the register to expr
A expr statement
A expr
                    Assemble one statement (instruction) at expr.
BR
                    Display the breakpoints currently set.
BR exprlist
                    Set each breakpoint in exprlist.
CA expr
                 Call a LisaBug subroutine
                Clear all breakpoints and patchpoints
Clear each breakpoint or patchpoint in exprlist
Display the value of each expression in hex and decimal.
CL
CL exprlist
CV exprlist
DB expr
                  Display memory as bytes.
DC expr
                    Disk Clamp.
DL expr
                  Display memory as long words.
DM exprl expr2
                    Display memory.
                    Set the SEG1/SEG2 bits.
DO expr
DR
                    Display index or ranges of dump RAM.
DU expr
                    Disk unclamp.
Dw expr
                    Display memory as words.
FB starting addr count data
                                        Find Byte.
                                      Find Long
FL starting addr count data
FM starting addr count data
                                       Find Memory
FW starting_addr count data
                                        Find Word
                    Start running at the current PC
G expr
                    Starting running at expr
ID
                    Disassemble one line at the next address
ID expr
                   Disassemble one line at expr
IL
                 Disassemble 20 lines at the next address
IL expr
                  Disassemble 20 lines starting at expr
IL exprl expr2 Disassemble expr2 lines starting at exprl
LP expr
                    Convert logical address to physical address.
MM exprl expr2
                    Display MMU information
MM num org lim ctrl Set MMU information
MR
                    Set a value level #5 interrupt on a word change.
P expr
                    Set port number to expr.
PA insrtaddr destaddr Insert a Patch.
PA
                    Display patch addresses
Q
                    Exit LisaBug, if it was called from Talk
RB
                    Reboot.
RM
                    Return to the Monitor.
                    Display the patch Return address Stack
SB exprl exprlist Set memory in bytes with exprlist starting at exprl
SC expr
                    Stack Crawl.
SD
                    Set the default radix to decimal
SH
                    Set the default radix to hex
SL exprl exprlist Set memory in long words with exprlist starting at exprl.
SM exprl exprlist Set memory with exprlist starting at exprl.
SW exprl exprlist Set memory in words with exprlist starting at exprl
SY
                   Display the values of all symbols
SY name
                Display the value of the symbol name
SY name expr
                   Assign expr to the symbol name
                    Trace one instruction at the current PC
T expr
                    Trace one instruction at expr
                  Display the Trace Display at the current PC
WP 0 or 1
                 Diable (0) or Enable (1) Write Protection.
```

INTRINSIC UNIT INSTRUCTIONS

LisaBug recognizes the Intrinsic Unit Instructions IUJSR, IUJMP, IULEA, and IUPEA. These instructions use the Line 1010 exception handler, and are created by the Linker. Full details can be found in the Development System Internal Documentation.

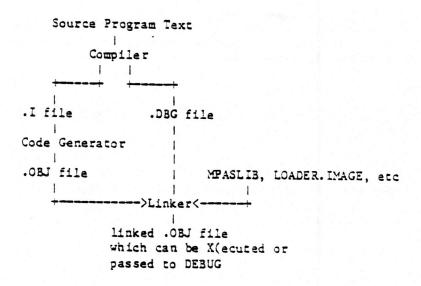
THE SYMBOLIC DEBUGGER

File Needed: DEBUGGER.OBJ

(the debugger is currently broken)

The debugger allows the programmer to follow the execution of a Pascal program, set traces and break points, and display or change variables without the need to insert cumbersome diagnostic statements. The basic unit of program execution is the Pascal statement and the basic unit of program data is the variable. If the debugger is present, run time errors drop the program into the debugger.

To use the debugger, first create a .DBG file in the compiler, then link the .DBG file with the output of the code generator, MPASLIB, and any other units that are needed. The .OBJ file that results can be executed normally, or you can X(ecute DEBUGGER and hand it the .OBJ file. The D(ebug option in the Monitor command line also invokes the Debugger.



Once executed, DEBUG takes the .OBJ file and asks you for a command. The first "command:" prompt occurs just after the start up of the program being debugged. The available commands are:

Set <proc [:offset] | address>

Set a breakpoint at the location given. PROC is the name of a procedure or function. The offset, if given, specifies the byte offset from the beginning of the procedure within the object code of the procedure. You can disassemble the procedure code and compare it with the source to decide where to set the breakpoint. ADDRESS is the absolute location at which the breakpoint is set. All numbers in the debugger are in decimal radix unless prefixed with 'S' (for hexadecimal). The address given

must be in core and within the code image.

Set Value (VarID | Address [size]>

Set Value sets a value breakpoint at the address given, or at the address of the variable identifier given. VarID can be either an entire variable, or just a field of it. When a check point is encountered therafter, the debugger checks all the value breaks. If the value of the variable or address has changed, the debugger stops, prints the old and new values of the variable or address in question, and waits for the next command. If you Set Value p' (which sets a value breakpoint on the variable pointed to by p), do not change the value of p. You cannot set a value breakpoint on a field of a packed variable.

Set (VarID)

Deposit a value in VarID. The debugger waits for the value to be typed. To retain the old value, type <cr>
 To set the value of a string variable, just type the new value (without quotes). It is not possible to set a string variable to the null string. String variables cannot be subscripted.

Set (Reg) (Num)

Deposit a value into one of the 18 registers (RDO-RD7, RAO-RA7, RPC, and RSR).

Set ..

Set "array abbreviation" mode. While in this mode, the present values in any array that is not a packed array of char are presented as first value, second value,"...", last value.

Clear ..

Clear .. gets you out of "array abbreviation" mode.

Clear

Clear clears all breakpoint and trace settings. You cannot clear the breakpoint which caused you to drop into the debugger.

Clear <Proc [:offset]>

Clears the breakpoint set by Set <Proc [:offset]>.

Clear Value

Clear Value clears all value breakpoints and traces.

Clear Value (VarID | Address [Size]>

Clear Value removes the breakpoint specified.

List Break

List Break lists the current break and trace points.

List (Proc [:offset] | Address>

List disassembles 20 lines of code starting at the address given. Sometimes fewer than 20 lines are disassembled. To continue the disassembly from this point, type List (without any arguments).

Regs

Regs lists the current values of all the user registers in hexadecimal.

Trace

Trace puts a trace on all procedure entries, but does not display procedure parameters.

Trace Proc

Trace Proc traces all procedure and function entries, displaying the parameters.

Trace Proc (Proc [:offset] | Address>

When an argument is given to Trace Proc, a trace is set in that procedure at the offset given. The parameters to the procedure are also displayed.

Trace (Proc [:offset] | Address) also puts a trace on the address given. The parameters to the procedure, hower, are not displayed.

Trace Value (VarID)

Trace value reports changes in the value of <VarID>, but does not stop execution.

Crawl

Crawl allows you to crawl up the stack as far as you like. Crawl starts at the current PC. At each stack location Crawl prints the current value, then waits for a carriage return. Any response other than <cr>
 exits Crawl and returns to the command request of the Debugger.

MacsBug

The Macsbug command drops you into Lisabug (the low level debugger). When entered in this manner, you are still in the debugger's environment. To get into Lisabug in the program's environment, first get the current PC of the program (from the Debugger), set a

Lisabug breakpoint at that location (BR or G TILL), then return to the debugger. Give the debugger the GO command, and when the program is started back up, you are dropped into Lisabug in the correct environment.

Check (Proc [:offset] | Address>

The debugger needs to be told when to check value breakpoints and traces. Check sets the location at which you want these checks to be made.

Go

Go restarts the program being debugged.

Quit

Quit causes the debugger to return you to the Monitor.

You can also poke around inside your code with the debugger. If you respond to the Command request with a name or a number, DEBUG responds as follows:

Name's Type	Action
Procedure or Function	Display segment number and offset within segment of that procedure or function, and the routine's in-code address, if any.
Variable identifier	Display the current value of the variable. The variable can (but need not) be qualified by field selection, indexing, or subscripting. Subscripts can be either integer constants, or a variable of the appropriate type.
Number	Convert hex to decimal, or vice versa, and display 16 bytes starting at that memory location. If there is an equivalent <pre></pre>
<number> TYPEID</number>	Print memory starting at <number> as if it were a variable of type TYPEID.</number>

You should not use any identifiers in your program that conflict with DEBUG command names. If your program terminates normally during debugging, DEBUG also quits.

If DEBUG does not understand a command, it merely asks you again for a command. In general, if DEBUG does not say it has done something, then it has taken no action at all.

To interrupt a program and drop into the debugger at almost any point, hit the NMI button on the side of the machine. In certain situations NMI may drop you into LisaBug instead of the debugger.

At link time, remember to include the .DBG extension when loading the debug file into the .OBJ file. The .DBG file must follow the .OBJ file to which it corresponds in the link. All separate compilation units must precede the main program in the link. If you want to use the .DBG files, you cannot do a partial link. The .DBG files and the resulting debuggable .OBJ files are significantly larger than the corresponding .OBJ files. The Debugger takes up about 10K of RAM plus about 60 words per procedure.

. THE FILER

File Needed: FILER.OBJ

INTRODUCTION

The Filer is modeled after the UCSD P-system's Filer and provides a similar set of functions. However, there are some small but important differences between the two.

The Monitor Filer requires that a colon follow a volume name in every case. It provides access to as many as 20 on-line volumes. The maximum number of files in a volume directory is 77.

All "workfile" commands and workfile-oriented features of the UCSD Filer have been omitted from the Monitor Filer. The functions of the Monitor utility programs Flipdir and Verify are provided by the Filer commands "S(ex" and "V(erify," respectively.

The UCSD Filer's "V(olume" command has been changed to "O(n-line" in the Monitor Filer. The UCSD Filer's "eX(amine" command is not available in the Monitor Filer.

The Monitor Filer's "T(ransfer" command performs automatic verification of all transfers between blocked devices.

ESCAPE aborts the currently executing function. When a wildcard R(emove or C(hange is aborted, you are asked whether to update the directory. A response of ESCAPE to this question is interpreted as 'No'.

The Monitor Filer includes a volume manager subsystem that permits you to maintain and manage the volume population on a Corvus drive. This subsystem, accessible through the "M" command, replaces the old VMGR utility.

FILER COMMANDS

The following is a list of commands that are recognized by the Filer. Filer commands are invoked by pressing the key which corresponds to the first letter of the command name.

B(ad-blocks - Scans for and reports bad blocks on blocked device.

C(hange - Changes a volume or file name on a blocked device. "Wildcard" file name specifications are recognized.

D(ate - Sets or changes the system date.

E(xtended directory

- Provides a detailed list of the contents of a blocked volume. "Wildcard" file names specify the display of a directory subset. You can write a directory to a printer with E #4:, PRINTER:

K(runch

- Creates the largest possible block(s) of contiguous space on a blocked volume by relocating existing files on that volume. It's a good idea to scan a volume for bad blocks before any attempts are made to Krunch it. Do not Krunch a volume that has bad blocks.

L(ist

directory - Provides an summary of the contents of a blocked volume.

See "E(xtended directory listing," above.

N(ew

- Creates a directory entry with the specified file name. Any volume name used to prefix the file name must be that of an on-line, blocked device. You can attach a size-specification suffix to the end of the file name. This suffix indicates the number of blocks to be occupied by the new file. The suffix consists of a non-negative integer constant or an asterisk ("*"), enclosed in square brackets ("[]"). For example,

FARLEY: MYFILE. TEXT[40] XRAY.OBJ[*]

The new file is placed on the specified volume in the first empty space that is large enough to hold it. The asterisk indicates that the file should fill half the largest free area on the volume, or all of the second—largest area, whichever is larger. In the absence of a size specification, the newly-created file occupies the largest area of contiguous free blocks on the volume. Files created with N(ew are stamped with the current system date, while the storage areas to which they correspond are left unaltered. N(ew permits the creation of zero-length files.

O(n-line - Provides a list of all volumes that are on-line.

P(refix - Changes the system prefix volume name.

Q(uit - Exits the Filer.

R(emove - Deletes entries from the directory of a blocked volume on a single-file or "wildcard" basis.

S(ex - Performs sexual reassignment of a blocked volume's directory. This command corresponds to the FLIPDIR utility. The Lisa is a female machine, whereas the Apple II is a male machine.

T(ransfer - Copies and transfers information between volumes.

Single-file or multiple-file wildcard transfers are allowed. You can also transfer between blocked and unblocked volumes. Transfers between blocked volumes are automatically verified, but transfers involving unblocked volumes are not.

- V(erify Compares blocked files for equality. You can compare single-files or multiple files common to two blocked volumes. Wildcard specifications can be used to name the comparands, so subsets of the files on one volume can be compared with a congruent subset of files on another volume. Verify detects and reports the following situations:
 - * The "source" and "destination" files match;
 - * The source differs from the destination in date-stamp, size, and/or contents (contents are always compared if sizes match, whether or not dates match);
 - * No counterpart to a given source file exists on the destination volume.

The report produced by Verify can be redirected to a device or file other than the console by following the destination file/volume name specification with a comma, then the name of the desired output device or file. For example,

Verify what file/vol ? VOL1:, VOL2:, PRINTER:

is equivalent to:

Verify what file/vol ? VOL1: Against what file/vol ? VOL2:,PRINTER:

The verification report in either of these cases is diverted to the PRINTER: device.

Erases and initializes the directory area of a blocked volume. If the volume already has a directory prior to the Z(ero, you have the option of retaining the old volume name and/or volume size. Z(ero can be used to increase or decrease the size of the virtual volume MEMORY:. Caution should be exercised, however, because it is possible to specify a volume size that is much larger than the LISA memory complement permits. In this case, a "memory overflow" is reported, and you should again invoke Z(ero to shrink MEMORY: to a reasonable size. Do not leave the Filer or attempt to use MEMORY: after receiving the "memory overflow" message!

Remember that Z(ero produces an empty directory. Therefore, to change the size of MEMORY: without erasing the directory, you must still use the CHANGEMEM utility.

vM(gr - Enters the volume manager (vMgr) subsystem, which presents its own sub-menu, and offers the following commands:

L(ist - List the hard disk Volumes (like Filer's O(n-line command).

From time to time, you may destroy the directory of one or more volumes that reside on the hard disk. The vMgr

subsystem assigns temporary names to these "bad" volumes so that you can be warned of their contamination, and can also manipulate them, if necessary. The form of such temporary names is BAD*n, where n is an integer (e.g., BAD*1, BAD*10, etc). Temporary names for "bad" volumes are effective only within the vMgr subsystem.

- M(ount M(ount assigns a hard disk volume to a specific Monitor device number, taken from the set [4,5, 9..20]. You can specify the device number to which a volume is associated, or you can accept the default selected by the vMgr. When vMgr picks a default unit number, it chooses the highest number
- N(ew N(ew creates new volumes. You can accept the default values for volume size and location as offered by the vMgr, or specify your own.

that is not currently in use.

- Q(uit Leave vMgr subsystem. If you have made any changes to the volume table you must confirm whether or not they should be made permanent in the default mount table. If you respond with any character other than 'Y', any changes made are temporary -- when the system is rebooted, the original settings will take effect.
- R(emove R(emove unmounts and destroys a volume. You can R(emove a M(ounted volume, but to do so you must approve the U)nmounting of that volume.
- U(nmount U(nmount is comparable to removing a floppy from a drive. It disassociates the volume from the unit on which it was mounted. The U(nmounted volume and the data it contains still exist on the hard disk drive, but can not be accessed through any Monitor device.

W(rite-

protect - W(rite-protect toggles the write-protection status for a volume. The contents of a write-protected volume cannot be changed. This command changes the default mount table.

Newly-created volumes are not write-protected.

See the Apple Pascal Operating System Reference Manual for further information.

PROFILE OR CORVUS INSTALLATION

To connect a Corvus or Profile hard disk to your Lisa:

 Turn everything off. You need two floppy disk drives, a controller card, and the latest monitor release with both the male and female boot volumes.

Set up the Apple so that drives #4 and #5 are available. Do not connect the hard disk yet.

Place the male boot volume in #4: and the female boot volume in #5:.
Turn on the Apple, the Lisa, and the hard disk. After awhile, the
Monitor command line should appear on the Lisa.

Plug the hard disk cable into the parallel port on the back of the Lisa. If the monitor command line starts flickering, type RETURN.

- Execute NEWZERO and respond 'Y'. NEWZERO initializes the hard disk mount and volume tables.
- 4. Execute the Filer and enter the volume Manager (type M).
- 5. Create (N(ew) a volume (MBOOT:) with 2048 blocks of space. Mount it as #20: (or anything other than #5:). Q(uit the Volume Manager and update the mount table (type 'Y').
- 6. Transfer all of the files on the female boot diskette (#5:) to the new volume (#20:). T #5:=, #20:\$
- 7. Enter the Volume Manager again and mount the new volume as #5:.
 Q(uit the volume manager and update the mount table.
- 9. Turn everything off. Set up the floppy disk drives as #9: and #4: (if you have two disk controller cards). Place the male boot volume in #4:. Turn on the Apple, the Lisa, and the hard disk.
- 10. Copy the entire monitor release onto the female boot volume (#5:), but do not overwrite any files that are already on #5:.

THE EDITOR

File needed: EDITOR.OBJ

LISA: EDITOR. FONT LISA: EDITOR. MENUS LISA: SYSTEM. FONT

INTRODUCTION

The mouse oriented editor is invoked by the monitor command E. Unlike the UCSD editor (invoked by U), this editor adheres to the Lisa User Interface. When invoked, it displays its menu, a portion of the Scrap folder, and a dialog box which asks you for the name of the file to be edited:

Get Document named?

Type the name of the desired file, followed by <RETURN>. The editor opens a folder and displays the first portion of the file. To open an empty folder (to start a new file), type just <RETURN> to the request for a document name.

The arrow or I-beam shows the current position of the mouse. The blinking vertical bar marks the insertion point. Activity takes place at the insertion point even if that point is not visible. If, for example, you open a folder, scroll to the end of the file, then start typing, the characters you type are inserted at the start of the file (where the cursor is), rather than at the end of the file (which you are merely looking at).

To mark text to be deleted or copied, set the insertion point to the start of the text (move the mouse there and click), then drag the mouse through the text to be acted upon. Selected text is displayed in inverse video. Click twice to select a word, three times to select an entire line. To select large pieces of text, put the cursor at the start of the text, move the mouse to the end of the text, and shift click at that point.

At any time you can,

Open a new folder

(select the PULL item in the DESKTOP menu)

Start editing in any folder on the screen

(select the desired folder from the tray icon menu, or click in the body of the desired folder)

Move the folder around on the screen (drag the folder's tab)

Make the folder larger or smaller (drag the grow box. The grow box is the square in the lower right corner of the folder)

Scroll up a line
(click the up arrow box in the lower right corner. To scroll continuously, hold the button down in the box)

- Scroll down a line

 (click the down arrow box in the upper right corner. To scroll continuously, hold the button down in the box)
- Jump back a windowful
 (click in the grey area above the elevator. Hold the
 button down in this area to continue flipping pages.
 The elevator is the empty box in the vertical scroll
 bar)
- Jump forward a windowful

 (click in the grey area below the elevator. Hold the button down to continue flipping pages)
- Jump to certain place in the folder (drag the elevator to the position in the scroll bar that corresponds roughly to the desired position in the file)
- Cut out the selected text and place it in the Scrap (select the CUT item in the EDIT menu, or type Command-Z)
- Paste the Scrap contents into the folder at the selection point (select PASTE in the EDIT menu, or type Command-X)
- Copy the selected text into the Scrap (select COPY in the EDIT menu, or type Command-C)
- Adjust the selected text right one space (select ADJUST RIGHT in the EDIT Menu, or type Command-R)
- Adjust the selected text left one space (select ADJUST LEFT in the EDIT Menu, or type Command-L)
- Save all your edits and close the folder (select PUT BACK in the DESKTOP menu)
- Save all your edits, but remain in the folder (select ACCEPT ALL EDITS in the DESKTOP menu)
- Write the current folder contents to another file
 (select CROSSFILE TO... in the EDIT Menu. CrossFile asks
 you for the file name to cross file to. If that file
 already exists, you are given a chance to change your
 mind before the old file is overwritten. CrossFile does
 not change the file name used by Accept All Edits or
 Put Back. If you do not want to crossfile after all,
 type (RETURN) as the filename).
- Cancel all the editing done since the last save command
 (Select UNDO ALL EDITS from the DESKTOP menu. The editor
 gives you a chance to change your mind before it cancels
 all your edits).

Exit from the Editor

(Select EXIT EDITOR from the DESKTOP menu. If there are unsaved edits in the folder, the editor asks you if these should be thrown away. The prompts force you to answer "Y" then "N" or vice versa to be able to get out, which is less than friendly.)

Set tab stops

(select SET TABS... from the EDIT Menu. You can change the number of spaces between tab stops. The default is eight)

Find some target string starting from the current selection (select FIND... from the SEARCH Menu. The default search ignores case and is token oriented. To change either of these, select the appropriate item in the SEARCH menu. FIND asks you for the target string. To find the same thing again, select FIND SAME. FIND & PASTE ALL performs a global find and replace. FIND can be invoked by Command-F, and FIND SAME can be invoked by Command-S. Only the first eight characters of a token-oriented search target are significant).

To move text from one folder to another, select and COPY the text from the source folder, activate the destination folder, set the cursor to the desired insertion point, and select PASTE.

CUSTOMIZING THE EDITOR

The editor uses whatever font it finds in the file LISA: EDITOR. FONT to display the folder contents. The suggested fonts are:

TITLE12R12S.F	20 lines x 82 chars
SARA8.F	26 lines x 83 chars
TILEX.F	32 lines x 82 chars (default)
TILE7R15S.F	32 lines x 94 chars
TILESRI8S.F	37 lines x 132 chars

THE UCSD EDITOR

File needed: UCSDEDITOR.OBJ

INTRODUCTION

The Editor is divided into the modes listed in the Editor's command line. A mode is invoked by typing the first letter of its name. Many of these modes also have command lines to further specify what action the Editor is to take. To insert new text, for example, you must type I when at the top level. Once in Insert mode, text can be inserted at that point in the file simply by typing it in. To accept the insertion type Ctrl-C (ENTER). To forget the insertion, type <ESCAPE> (the CLEAR key in the numeric keypad). To get to another mode (to move to a different place in the file, for example), you must first exit the mode you are in.

The entire file being edited must be in memory at all times during editing. There is no macro facility. The cursor is sometimes not where it appears to be. If a key is held down for more than a fraction of a second, the key's action is repeated at about 10 times per second.

Elaborate documentation can be found in the Apple /// Introduction, Filer, and Editor Manual or in the Apple][Pascal Operating System Reference Manual. The Apple][manual is somewhat inaccurate because our Editor is an enhanced version of the Pascal 1.1 update of the original Editor.

CURSOR MOVEMENT

Numerical Arguments

Most commands can be preceded by a numerical argument. The argument can be an integer between 0 and 9999, or the character / which means "as many times as possible". The numerical argument defaults to 1. It is abbreviated as "n" below.

Set Direction

The Set direction is the direction in which some commands move through the file. The first character in the command line tells what the current direction is:

> is forward, < is backward.

At the top level of the Editor, the keys

> . +

change the direction to forward, and

< ,
change it to backward.

Simple Cursor movement

(SPACE) move n spaces in set-direction (RETURN) move n lines in set-direction (you end up at the start of the line) Up-Ariow move n lines up Down-Arrow move n lines down Left-Arrow move n spaces left Right-Arrow move n spaces right move n tab positions in set-direction (TAB) move to start of last text replaced, found, <s>. or inserted move to start of line move to end of line \$

W move to beginning of next word
E move to end of current word
B move backward one word

T move to the next occurrence of a specified character

You can also use the main keyboard "h", "j", "k", and "l" keys for cursor movement:

h move left
j move down
k move up
l move right

G(oto

Jump to some place in the file. You are presented with a secondary command line to tell the cursor where to Jump:

B(eginning of file E(nd of File M(arker

Jump to a specific marker. You can have up to 10 active markers set by the Set command (see below). Markers are pointers to file, not text, locations. You can return to the point you jumped from if you immediately type <ESCAPE>.

P(oint

like <=> above.

<ESCAPE> exits without Jumping.

P(age

Move the cursor n windows in the set-direction.

F(ind

Find a target string and place the cursor just after it. The normal mode is case sensitive. With the exceptions of S, L, U, and T, the first character following the F (for FIND) is the delimiting character. The target string specification ends when the delimiting character is repeated. Thus F.hi. looks for one

occurrence of the lowercase word "hi". /F(HI(places the cursor after the last occurrence of the uppercase word "HI".

L(iteral, T(oken

FIND can be told to look for any occurrence of the target string (Literal mode), or only for delimited occurrences (Token mode). The default is the one not given as an option when you type F (usually Token mode is the default). To override the default for one search, type L or T after F, but before the delimiting character. FL.hi. looks for any occurrence of the two characters "hi" in the file. A delimiter (for Token mode) is any character which is not a letter or a number. See S(et E(nvironment below to change the default mode. Note also that FL/<RETURN><RETURN>/ finds the next blank line.

U(pper-and-Lower-Case

To get a case insensitive search, type U before the delimiting character. FU.hi. looks for any occurrence of "HI", "hi", "Hi", or "hI" in the file.

S(ame-String

To repeat a search using the same target string as was used in the last search, you can type S in place of the delimited string. The Environment information (see S(et E(nvironment) tells you the current target string. If we last searched F.hi., FS searches again for "hi". Mode changing letters can be concatenated: FLUS looks for the target string last specified, in Literal mode without sensitivity to upper or lower case.

(ESCAPE)

exit Find mode without taking any action.

TEXTUAL COMMANDS

I(nsert

Insert text into a file. Insertion takes place between the character on which the cursor is sitting and the character immediately to its left. Ctrl-C accepts an insertion, <ESCAPE> forgets it. At any point during an insertion, left arrow deletes the character to the left of the cursor. You can't backspace past the beginning of the insertion. If you accidentally hit <ESCAPE> during an insertion (and thereby delete everything you just typed), you can rescue the inserted text by C(opying it from the B(uffer.

D(elete

Delete text moving in the Set-direction. Ctrl-C accepts a deletion, <ESCAPE> restores the deleted text. All deleted text is stored in the Copy Buffer, so to move a piece of text from one place to another, first delete it, move the cursor to the new location, then C(opy from the B(uffer. D(elete accepts numerical arguments (including / and P(age).

Z(ap

Delete all text between the current cursor position and the current location of the "Point" (the location of the last text inserted, found, or replaced). Zapped text is placed in the Copy Buffer.

C(opy

Copy text into the file at the current cursor position. The copied text can be taken from the Copy Buffer, or from a file.

F(ile

Copy a file, or some portion of that file. If the file has markers placed in it, you can choose to copy only the portion between any two markers by enclosing the names of the markers in square brackets after the file name. HI.TEXT[PROC1,PROC3] copies the portion of HI.TEXT lying between the markers PROC1 and PROC3. [,MARKER] copies from the beginning of the file to MARKER, and [MARKER,] copies from MARKER to the end of the file. If no marker specification is given, the entire file is copied.

B(uffer

Copy the contents of the Copy Buffer.

M(arker

Copy text between two markers, using exactly the same syntax as that used in copying from a file.

(ESCAPE)

Exit Copy mode without copying anything.

X(Change

Exchange mode allows you to overwrite text (exchange old text for new). You cannot exchange characters past a <RETURN>. Ctrl-C accepts an exchange, <ESCAPE> forgets it. Right arrow copies characters as it passes over them.

R(eplace

R(eplace mode is very similar in syntax to the F(ind mode. R(eplace takes two strings—the target string (as in F(ind), and the string to substitute for an occurrence of the target. Numerical arguments are valid.

L(iteral, T(oken

See F(ind above.

U(pper-and-Lower-Case

See F(ind above.

S(ame-String

See F(ind above. Note that S can be used for either or both strings: RSS looks for the previous target string and replaces it with the previous substitute string.

V(erify

If the find-and-replace operation is done in V(erify mode, you are asked before each replacement whether the replacement should take place for the given occurrence of the target. /RV.hi./ho/ looks in the set-direction for every occurrence of "hi", then on each one asks you if you really want to replace that "hi" with "ho".

KESCAPE>

Exit Replace mode without replacing anything.

A(djust

Adjust the indentation of a line, or a group of lines. Left and Right Arrow push the entire line left or right. (SPACE) moves a line a space in the set direction. Up and Down Arrow cause the same adjustment to affect lines above or below the first adjusted line. (RETURN) affects the lines above or below, depending on the set direction. (TAB) moves lines over 8 spaces.

Alternate choices are:

L(eft-justify R(ight-justify C(enter P(age

Ctrl-C accepts the adjustment-you have no other choice. Numerical arguments (including /) can be used.

M(argin

The M(argin command attempts to fit a paragraph into the prevailing paragraph formatting parameters set in the E(nvironment. M(argin only works when I(ndent-Auto is false and F(illing is true. A paragraph is defined to be any text bounded above and below by two delimiters. The allowed delimiters are a blank line, the beginning of the file, the end of the file, and a line which begins with the Command Character. To M(argin a paragraph, move the cursor to any point within the paragraph and type M.

O(bscure

O means "join" for some reason. It replaces the <cr> at the end of the current line with a space, thereby joining the two lines.

EDITOR DIRECTIVES

S(et

S(et mode allows you to set Markers and to change various global Editor settings.

M(arker

A Marker is an absolute location in a file to which you can J(ump. A maximum of ten markers can be active at one time in a file. To set a marker, move-the cursor to the desired location, then type SM. You are then asked for the marker's name (a character string of up to 8 characters).

E(nvironment

The editing "environment" can be customized to suit your whims. The E(nvironment command displays the current settings of various parameters, the current size of the file, the current target and replacement strings for F(ind and R(eplace, the markers currently in the file, and various other information.

To change the setting of the parameters, type the first letter, then the new value. SEAF, for example, enters S(et mode, enters the E(nvironment menu, then sets A(uto-Indent to False. To exit the Environment menu and retain the changed values, type Ctrl-C (ENTER).

A(scii-file

Used by BASIC on Apple ///. Not applicable to Lisa.

C(ommand Character

You can set the Editor Command Character to be any printing character. The command character can be used to protect a line from the Margin command.

F(illing

If Filling is true, lines are automatically broken at the right margin between words.

I(ndent-Auto

If I(ndent-Auto is true, a new line is automatically indented to the position of the first non-space character of the previous line.

L(eft, R(ight, P(aragraph margin

These three parameters control the layout of the paragraphs when using Filling or Margin. P(aragraph margin is the indentation of the first line of the paragraph.

N(ame-of-file

When you Q(uit the Editor, the S(ave option can be used to write the edited file to the file named in

the N(ame field. You can change this name to any legal file name.

S(pace-continue

If TRUE, a space must be typed before you can continue editing after an error has occurred. If FALSE, any character can be typed to allow you to continue.

T(oken

T(oken defines whether the F(ind and R(eplace commands use the T(oken or L(iteral mode as a default.

(ESCAPE)

Restore all the parameter values that were in force when Set mode was entered, and exit Set mode.

V(erify

V(erify causes the Editor to redisplay the screen.

Q(uit

Exit the Editor or load a new file.

U(pdate

Update the "workfile". It is not an accident that this is the only mention of the "workfile".

E(xit

Exit the Editor without saving anything that you did.

R(eturn

Return to the Editor (in case you didn't want to quit).

S(ave

Save the current edited file under the name given when the Editor was entered, or (if you started a new file) under the name last given in a W(rite command. Return to $Q(uit\ menu.$

W(rite

Write the current edited file to a diskette file. You must give the file's name. W(rite returns to Q(uit menu.

C(hange

Load a new file into the Editor without restarting the Editor. The previous target and substitution strings are carried over, and the copy buffer is maintained intact (if possible).

UTILITY PROGRAMS

(IUManager, ChangeSeg, SegMap)	•	 •	72
(Configure, Contrast, SetSP, ChangeMem, Flip4, MoveSoroc)		 •	75
(FileDiv, FileJoin)			77
(Diff, FindID, Pretty List, PascalRef)			80
(DumpObj, DumpHex, Patch, ObjDiff, ByteDiff, GxRef)		 •	87
(LisaTest)	•	 •	91
(Perform, Coverage Analysis)			93
(Script)	•		96
(Terminal Emulator).	_		9,8

IUMANAGER

IUMANAGER modifies the file INTRINSIC.LIB used by the intrinsic unit Linker and loader to find the intrinsic unit files. INTRINSIC.LIB is essentially a directory of unit names, segment names, and file names. When executed, IUMANAGER asks for the input and output files to be modified. The default name for both files is *INTRINSIC.LIB. The intrinsic unit manager has three modes: Manage Segments, Manage Units, and Manage Files. Each mode operates on an associated table of information used by the loader to properly link intrinsic unit code and data into an executing program.

Manage Segments Mode operates on the Segment Table which contains a list of segment names with information about each segment for the loader and linker. Manage Units operates on the Unit table which contains the unit names and information used by the loader and linker to build the data pointer table. Finally, Manage Files operates on the File Table which contains a list of files indexed by a file number. The loader uses the file number to find the intrinsic units and segments on the disk.

IUMANAGER has the following commands:

Q(uit Write the output file and exit from IUMANAGER.

S(egments Enter the Segment Manager and list the contents of the Segment Table.

U(nits Enter the Unit Manager and list the contents of the Unit Table.

F(iles Enter the File Manager and list the contents of the File Table.

In each of the modes you can:

L(ist List the contents of the currently active table. If you have more than 32 entries in the table, you can stop the listing with Control-S (the '* key on the numeric keyboard).

R(emove an entry from the currently active table.

N(ew Create an entry in the currently active table.

The New command prompts you the value of each field.

A response of <cr>
accepts the default value.

In the File Manager you have one further command choice:

I(nstall (update) the segment and unit tables from a linked object file. The Install command prompts you for the file number of the entry to be updated.

CHANGESEG

CHANGESEG changes the segment name in the modules in an object file. The first prompt asks for the object file you want to change:

File to change:

Changes are made in place (the file itself is changed). You are next asked:

Map all Names (Y/N)

If you want to change segment names in all modules, respond Y. If you want to be prompted for the new segment name for each module, type N. A response of $\langle \text{cr} \rangle$ accepts the default name.

SEGMAP

SEGMAP produces a segment map of one or more object files. The first prompt:

Files to Map ?

accepts either an object file name or a command file name. A command file must be preceded with a <. SEGMAP adds the .TEXT suffix to the command file name. The next prompt:

Listing File ?

directs the map information to the file given. A response of #1: or CONSOLE:, for example, send the map information to the screen. The map information includes the object file name, the name of the unit in the file, the names of the segments used in that unit (if any), and the new segment names.

CONFIGURE

CONFIGURE modifies some of the vectors in the Monitor Map Table. These vectors are stored in CONFIG.DATA on the male boot volume and are used by the Monitor to configure your system when it is booted. To use CONFIGURE, copy CONFIG.DATA from the male side to a female volume, or flip the sex of the boot diskette. X(ecute CONFIGURE. CONFIGURE asks you whether it should Go or Quit. Type G to run CONFIGURE, Q to return to the Monitor command line. CONFIGURE asks you for the file containing the vectors you want to change. If you do not give a volume name, it look on the prefix volume. If you give just the volume name, it looks for CONFIG.DATA on that volume. CONFIGURE can change the following vectors:

dE(bug pointer	[\$150]				
D(efault Stack pointer	[\$13C]	(*	most	important	*)
H(eap pointer	[\$138]				
C(orvus pointer	[\$134]				
U(art pointer	[\$110]				
A(pple port	[\$118]				
M(emory top	[\$114]				
S(creen base	[\$110]				
B(uffer pointer	[\$10C]				

The old and new value of each vectors is also displayed. Type the capitalized letter of the vector you want to change. Lower case is allowed in hexadecimal numbers. When you are done, type Q to Quit. At this point you are asked where to save the new values. You can write the changes back to CONFIG.DATA, exit without making any changes, and so on.

A memory map is given in the Monitor chapter showing the relationship of these vectors. Do not place the start of the heap above the stack pointer. Because CHANGEMEM sets aside heap space, it is safer to set the stack pointer before grabbing a lot of the heap with CHANGEMEM.

Once you have finished modifying CONFIG.DATA, transfer it back to the male boot volume so that it can take effect when the system is rebooted.

CONTRAST

CONTRAST changes the contrast setting of your screen without changing the default setting. It is a simple program that should be self-explanatory. Like CONFIGURE, it first asks whether to G(o or Q(uit. If you type G, some alphanumeric characters are scattered around the screen for reference. You can type '>' or '.' to increase and '<' or ',' to decrease the screen contrast.

SETSP

SETSP sets the address at which the stack pointer starts. Memory above that address is then reserved for code, and memory below it is the stack and heap space. If your program requires a great deal of room for data, set the stack pointer to a high address. If the program requires a great deal of code, set the SP to a low address. The Monitor default SP starting address depends on the version of CONFIG.DATA on your male boot volume. The highest possible address is the bottom of the Monitor.

All SETSP I/O is in hex. When X(ecuted, it displays the current stack pointer value. Type O to exit the program. The optimal value to give SETSP may not be obvious at first, since code swapping can change your memory requirements. It is quite possible that a program will run happily for hours, then die with a Loader Error when a piece of code couldn't be fit in memory. If this happens, set the stack pointer to a lower starting address, and try again.

CHANGEMEM

CHANGEMEM changes the size of the predeclared RAM-resident volume MEMORY:. Its interface is identical to that of SETSP. The default size of MEMORY: is 10 blocks. Space for the MEMORY: volume is taken from the available heap space.

FLIP4

Volume #4: is normally the RAM-based volume MEMORY:. The Disk drive that would usually be #4: is hidden from the monitor to avoid overwriting the male boot volume. If you want access to that disk drive from the monitor, run FLIP4. FLIP4 executes a simple loop until you tell it to Quit. After asking whether to continue or to quit, FLIP4 gives you a chance to toggle the state of #4:. #4: is either MEMORY: or the disk drive. Remember to remove the male boot volume before writing to #4:.

MOVESOROC

MOVESOROC (also sometimes known as MS) determines where the Pascal WRITELN output goes. It normally is sent to the Lisa screen. When an application is running on this screen, however, debugging WRITELNs mess up the program's pretty output. MOVESOROC redirects this output to either the Apple monitor, the UART (serial port #2), or back to the Lisa. Monitor input always comes from the terminal to which output has been directed.

FILEDIV and FILEJOIN

It is often necessary to distribute files that are too large to fit onto a single floppy diskette. FILEDIV can be used to break a large file into several diskette-sized pieces. FILEDIV can then be used to rejoin these pieces at the file's destination. These two programs replace the TRANSFER program.

To divide a large text or object file, execute FILEDIV.

Input file: <give the name of the file to be divided>
Output file: <give the name to be used for the output files>

Do not include the suffix in the file name. If, for example, you want to divide TEMP.TEXT, give TEMP as the input file, and TEMP (or whatever) as the output file. FILEDIV will create a group of files named TEMP.1.TEXT, TEMP.2.TEXT, and so on, until TEMP.TEXT is completely divided up. If you use the drive number (#9:, for example), rather than the volume name, the new files can be written to multiple diskettes. When space on a diskette is exhausted, FILEDIV asks you to insert another diskette.

To rejoin the pieces of the file, execute FILEJOIN. Using the example given above, we can rejoin TEMP.1.TEXT and friends into TEMP.TEXT by responding:

Input file: TEMP (will read TEMP.1.TEXT, etc)
Output file: TEMP (will create TEMP.TEXT)

FILEDIV and FILEJOIN use regular directories, so a spurious sex change cannot destroy your file. Files are verified in both directions.

DIFF

DIFF is a program for comparing ".TEXT" files, in the LISA Pascal development environment. DIFF is strongly oriented toward use with Pascal or Assembler source files.

DIFF is not sensitive to upper/lower case differences. All input is shifted to a uniform case before comparison is done. This is in conformance with the language processors, which ignore case differences.

DIFF is not sensitive to blanks. All blanks are skipped during comparison. This is a potential source of undetected changes, since some blanks are significant (in string constants, for instance). However, DIFF is insensitive to "trivial" changes, such as indentation adjustments, or insertion and deletion of spaces around operators.

DIFF does not accept a matching context which is "too small". The current threshold for accepting a match is 3 consecutive matches. The M option allows you to change this number. This has two effects:

Areas of the source where almost "every other line" has been changed will be reported as a single change block, rather than being broken into several small change blocks.

Areas of the source which are "entirely different" are not broken into different change blocks because of trivial similarities (such as blank lines, lines with only "begin" or "end", etc.)

DIFF makes a second pass through the input files, to report the changes detected, and to verify that matching hash codes actually represent matching lines. Any spurious match found during verification is reported as a "JACKPOT". The probability of a JACKPOT is very low, since two different lines must hash to the same code at a location in each file which extends the longest common subsequence, and in a matching context which is large enough to exceed the threshold for acceptance.

DIFF can handle files with up to 2000 lines.

DIFF first prompts you for two input file names: the "new" file, and the "old" file. DIFF appends ".TEXT" to these file names, if it is not present. DIFF then prompts you for a filename for the listing file. Type carriage-return to send the listing to the console.

DIFF does not (currently) know about INCLUDE files. However, DIFF does allow the processing of several pairs of files to be sent to the same listing file. Thus, when DIFF is finished with one pair of files, it prompts you for another pair of input files. To terminate DIFF, simply type carriage-return in response to the prompt for an input file name.

The output produced by DIFF consists of blocks of "changed" lines. Each block of changes is surrounded by a few lines of "context" to aid

in finding the lines in a hard-copy listing of the files.

There are three kinds of change blocks:

DELETION -- a block of lines in the "old" file which does not appear in the "new" file.

REPLACEMENT -- a block of lines in the "new" file which replaces a corresponding block of different lines in the old file.

Large blocks of changes are printed in summary fashion: a few lines at the beginning of the changes and a few lines at the end of the changes, with an indication of how many lines were skipped.

DIFF has three options which allow you to change the number of context lines displayed (+C), the number of lines required to constitute a match (+M), and the number of lines displayed at the beginning of a long block of differences (+D). To set one of these numbers, type the option name followed by the new number to the prompt for the first input file name. +D 100, for example, causes DIFF to print out up to 100 lines of a block of differences before using an ellipsis. The maximum number of context lines you can get is 8.

FINDID

FINDID searches code files for an identifier. It provides a service similar to that of the editor's literal search, but the search can cover any number of files of any size. When executed, it asks first for the name of the file which contains the list of files through which you want to search. For example, if you want to search the files CODE.TEXT, CODE1.TEXT, and CODE2.TEXT, make a file which contains:

Code.Text Codel.Text Code2.Text

and give FINDID this file's name. FINDID then asks for the identifier you want to search for. Only the first eight characters are significant. The search is always literal—any identifier beginning with the specified eight characters is considered a match. FINDID's last prompt asks whether the search should ignore case differences. FINDID then grovels through files in the list reporting any occurrences of the identifier. To get out of FINDID, hit NMI, then type RM to LisaBug.

PRETTY LIST

Pretty List scans a listing produced by the Assembler, and replaces the asterisks in the displacement portion of branch instructions with the actual forward reference value. When you X(ecute PRETTY, you are asked for the Input File (the Assembler listing file). Because this file can be either a data file or a text file (with a .TEXT extension), the next prompt is:

If input file is a text file (file.text) type 1 else type 0 --Pretty List then asks for the output file name.

If the listing file contains:

03601				CHKLO	BSR4	CHKMEM
03601	49FA	***	#		LEA	@1.A4
03641	6000	***	1		BRA	CHKMEM
0362*	0006					

Pretty list produces:

03601				CHKLO	BSR4	CHKMEM
03601	49FA	0006	#	•	LEA	@1.A4
03641	6000	005A	#		BRA	CHKMEM

PASCALREF

Pascalref is a cross reference utility for Lisa Pascal programs. It can perform partial or complete cross references, can handle USES and INCLUDE statements correctly, and imposes no limit on the size of the target program.

Pascalref assumes that the program or unit to be referenced (target program) has been compiled without syntax errors. It also assumes that the font BOLDIOV and the file MPMENUFILE.TEXT are available on your prefix volume or the boot volume (#5:).

THE USER INTERFACE

ACTION	SEARCHOPTIONS	FINDTYPES	YN-TF	<- The Menu	Bar
Setup Files	Interactive	Declared	Yes-True		
Set Scope	Reloffsets	Modified	No-False		
Begin Pascalref	Procdic Widepaper Used Unit Int	Accessed Stnd PFT			
	Out Scope Vars				

The line in capitals at the top represents the menus in the menu bar. The lower case names are the items in each pull down menu. A shaded menu item shows that the option represented by that item is active. To change the 'Procdic' option, for example, use the mouse to select 'Procdic' then select either True or False. You can also type SP, then enter Y(es or T(rue, N(o or F(alse. When you have all your options set up, select 'Begin Pascalref'.

OPTIONS

Setup Files

You are asked for the names of the listing file, source file, and output file.

Set Scope

Pascalref allows you to set the scope to be a single procedure. Only identifiers within that procedure and its local procedures are referenced. Of course, accesses to any variables global to the procedure are included in the OUT OF SCOPE section. The default scope is the whole program. Currently, when referencing a Unit by itself, only set the scope to the whole unit. Also, don't set the scope to a FORWARD procedure or a procedure in the interface of a unit.

Begin Pascalref

Start the cross reference.

SEARCHOPTIONS

Interactive --

When Interactive is true (the default), Pascalref looks only for those names that occur in the list of variables that you type in.

When a particular reference is finished, Pascalref asks you 'Look at more identifiers?'. If you answer yes, PascalRef returns to the options setting routine with the same files set up. You can change the options and the files on each pass if you want to.

When Interactive is false, Pascalref cross references all identifiers within the specified scope.

Reloffsets

Next to each occurrence of an identifier is the line number it occurs on. When Reloffsets is true (the default), the line numbers are listed relative to the procedure the occurrence is in.

When Reloffsets is false, offsets are given relative to the beginning of the program.

Procdic

If Procdic is true, Pascalref creates a procedure dictionary listing each Procedure with the line it starts on. The default for ProcDic is false. The program or outermost procedure in the scope is procedure #1. Nested procedures are indented.

In the case of forward procedures and procedures declared in the interface of a unit, the procedure number listed reflects where the procedure declaration occurs. '=Forward Proc' appears in the procedure dictionary after the 'STARTING LINE #'. The ordered location of the procedure name and the starting line number within the procedure dictionary reflect where the header to the body part of the procedure occurs.

Widepaper

The default value (False) should be used when sending output to the console or to standard 8-1/2 inch wide paper. When printing on 132 column paper, set Widepaper to true.

Used Unit Int

The default value of True causes the INTERFACE parts of USED units to be included in the PASCALREF scan. This allows you to see where every identifier used by a Pascal program is defined. If you are not interested in the INTERFACE of a unit, set 'Used Unit Int' to false.

Out Scope Vars

When a program accesses a identifier that has not been defined, that access shows up in the OUT OF SCOPE VARIABLES OR PROCEDURES section of the PASCALREF listing. This part of the listing is present when 'Out Scope Vars' is true (the default).

FINDTYPES

The three 'Find Types' are DECLARED, MODIFIED and ACCESSED. The default value of each type is true. Each type can be set independently. When all are true, all occurrences of an identifier are listed. If, for example, only ACCESSED is true, PascalRef lists only places where a variable is accessed. MODIFIED flags places where a variable occurs to the left of :=, and where it is passed as the actual parameter to a formal VAR parameter. DECLARED lists places where an identifier is declared.

Stnd PFT lists occurrences of standard procedures, functions and types. Its default is false.

YN-TF

This menu gives you an option for answering Yes-No and True-False questions. Choosing Yes-True is the same as entering a 'Y' or 'T' from the keyboard and choosing No-False is the same as entering a 'N' or 'F'.

OUTPUT FORMAT

LISTING (Pascalref can produce a listing identical to that produced by the compiler)

PROCEDURE DICTIONARY:

PROC #,	PROC NAME,	STARTING LINE #
1:	MAINPROGRAM	
3:	NESTEDPROC	0 80
4:	FURTHERNE STEDPROC	120
5:	PROC	200
2:	FORWARDPROCWHOSEBODYCOMESAFTERPROC	40=Forward Loc
6:	Procne sted inbody of forward proc	280
7:	LASTGLOBALPROC	300

The identifi	The procedu	res i	t occurs	in nces withi	n those p	rocedures	
IDENTIFIER	PROCEDURE	occ	URRENCE			ssed, M=m P=passed	odified, to Var param)
I	PROCA PROCB			45, A 20, A	50, A 30,	60.	
STRNG	PROCA	D A	12, M 60,	41, M	62, A	50, A	58,
	PROCC	V	75, M	80, A	82.		

OUT OF SCOPE VARIABLES OR PROCEDURES

These are listed in the same format as that of the regular identifiers, but represent items that are global to the chosen scope. To find out what global variables a procedure and its nested procedures access, set the scope to the procedure, set INTERACTIVE to false and look at the resulting OUT OF SCOPE items.

GENERAL NOTES ON THE USE OF PASCALREF

Pascalref does not store information about variable types or record structures. If you have both a stand alone variable named AVAR and a record field named AVAR, PascalRef lists both as the same identifier.

Include commands are recognized by Pascalref and the INTERFACE parts of units USED by the target program are included in the

reference if the scope is set to the whole program.

To find variables that are declared but no longer used by a program, do a reference of the whole program. Variables that have a 'D' occurrence and no others can often be removed from the program.

Occurrences of a particular identifier are always exactly in order when interactive is true. When interactive is false, occurrences are grouped by the procedures where the identifier is declared locally. In the case where interactive is false, you may notice the following:

I PROCA D 10, P 45, A 50, A 60 PROCB D 16, A 20, A 30,

The variable 'I' was declared and accessed in PROCA and declared and accessed in PROCB. The accesses in PROCA occur after the declaration and accesses in proc B but they are listed first.

If 'I' were not defined in PROCB, it would look like:

I PROCA D 10,
PROCB A 20, A 30,
PROCA P 45, A 50, A 60.

If the first example were done in interactive mode, it would look like:

PROCA D 10, PROCB D 16, A 20, A 30, PROCA P 45, A 50, A 60.

Procedures and Functions as parameters are currently not fully implemented in Pascalref. They are parsed by Pascalref, but Variables passed to a procedure or function that is a parameter are always marked as modified in that occurrence.

DUMPOEJ

DUMPOBJ is a disassembler for 68000 code. It can disassemble either an entire file, or specific modules (procedures) within the file. DUMPOBJ replaces DUMPMCODE.

DUMPOBJ first asks for the input file which should be an unlinked object file. The output (listing) file defaults to CONSOLE:. You are asked whether you want to dump

A(11, S(ome, or P(articular modules.

If you respond S(ome, DUMPOBJ asks you for confirmation before dumping each module. A response of <ESC> gets you back to the top level. If you respond P(articular, DUMPOBJ asks you for the particular module(s) you want dumped.

The next question is: 'Dump file positions [N]?' The file position is a number of the form [0,000] where the first digit is the block number (decimal) within the file and the second number is the byte number (hexadecimal) within the block at which the module starts. This information can be used in conjunction with the PATCH program. Finally, DUMPOBJ asks if you want the object code disassembled.

DUMPHEX

DumpHex provides a textual representation of the contents of any file. The file dump is block-oriented with the hexadecimal representation on the left and the corresponding ASCII representation on the right. If a byte cannot be converted to a printable character, a dot is substituted.

When DumpHex is X(ecuted, it asks you for the name of the output file. A .TEXT extension is added if necessary. To direct the output to the console, type carriage return. After getting a valid output file name, DumpHex asks for the input file to be dumped. No extensions are appended, so give the full filename. Once a file has been completely dumped, DumpHex asks you for the next file to dump. Type carriage return to exit the program.

After opening the input file, DumpHex asks you which block to dump. The default (carriage return) is block 0. If the output is going to a file, you are asked which block is the last you want dumped. The default here (carriage return) is the last block in the file.

The format of the console output depends on the number of lines your screen has. If fewer than 33 lines are available, the output is displayed only a half block at a time. Between blocks or block halves you have the option to

Type (space) to continue, (escape) to exit.

Escape returns to the prompt for an input file.

PATCH

Patch allows you to examine and change the contents of any file. The display of the file's contents is exactly like that of DumpHex. With Patch, however, you can use the cursor control keys to move around in the block and change the value of any byte using either the hexadecimal representation on the left or the ASCII representation on the right.

After X(ecuting Patch you are asked for the full name of the file to patch. Carriage return exits Patch. No extension is appended to the file name. You are then asked for the number of the block you want to mess around with. Carriage return here returns you to the file name prompt.

The block is displayed with the cursor in the upper left corner at word 0 of the block. The arrow keys can be used to move around in the block. If you move the cursor up from the top line, you get the bottom line of the preceding block. Similarly, if you move down from the bottom line, you move into the top line of the next block.

When the cursor is on the hexadecimal side of the display, you can change any byte by typing the new hexadecimal value. Any non-hex characters are ignored. You can impress your friends by pointing out that the change is reflected automatically in the ASCII portion of the display. When the cursor is on the ASCII side, type any character to replace the value of the byte. Until you move out of the block you can undo any changes by typing <escape>.

OBJDIFF

OBJDIFF performs a comparison of two object (.OBJ) files. The two files being compared should be very similar. OBJDIFF uses procedure boundaries to get itself back in sync after a difference is found.

BYTEDIFF

BYTEDIFF compares any binary files, but once it finds a difference between the two files, it does not always find where the differences end.

GXREF

GXREF lists all the modules which call a given procedure, and all the modules which that procedure calls. It provides a global cross reference of subroutines and modules.

LISATEST

LISATEST is a package of hardware test routines. The DLAG: diskette which contains these programs can be obtained from Rich Castro. To use the programs, boot the Apple II with the Lisa in the power-on reset state, then X(ecute LISATEST. You have eight choices:

- 1) Apple-Lisa Interface Test
- 2) Memory Test (RAMTEST)
- 3) Display Memory Test Results
- 4) UART Wrap Around Test
- 5) Video Latch Test
- 6) MMU Test
- 7) Keyboard Test
- 9) Quit

The tests are, for the most part, self-explanatory. For a complete description of each test, its prompts, error messages, and options please see the Lisa Production Tests documentation by Rich Castro. A short description of each test is given below.

Apple-Lisa Interface Test

The Interface Test attempts to use the parallel port interface between the Apple II and the Lisa to verify that the two systems can communicate with each other.

Memory Test

The Memory Test program tries to single out bad or marginal memory chips and provides trouble shooting information about other memory board problems. It is essentially an updated and easy to use version of RAMTEST.

Display Memory Test Results

After running the Memory Test, you can have the results of that test redisplayed by the Display Memory Test Results program.

UART Wrap Around Test

The UART Test checks the UART on the CPU board that controls the RS-232 port #1 (the one on the left as you face front of the machine). To run the test you need a specially wired wrap around DB-25 male connector.

Video Latch Test

The Video Latch Test checks the operation of the video page latch on the MCU board.

MMU Test

The MMU Test tries to verify that the MMU is working properly. It sets up the base and limit registers in the MMU with various values and then attempts to access the corresponding memory segments.

Keyboard Test

The Keyboard Test checks the keyboard and mouse buttons to verify that the COPS interfaces are functioning properly.

PERFORM

Perform monitors the execution performance of a program. After X(ecuting Perform, you are asked for the listing file's name. A carriage return directs output to the console. If necessary, .TEXT is appended to the listing file name. You are next asked for the name of the program you want to analyze. If necessary, the extension .OBJ is added to the file name. The program file must be executable and must be linked with the corresponding .DBG files.

Perform scans the program file for procedure entry points, listing them as they are found. It then waits for you to type a space before executing the program. Every 1/60 second the program's program counter is checked to find out which routine is executing at that moment. When the program terminates, Perform produces a listing of the routines it found executing ordered according to the amount of time spent in each routine. Routines that were never caught executing are listed separately. Perform may miss the execution of short or rarely called routines.

The longer the program runs, the more trustworthy the analysis. Routines that are synchronous with the 60 Hz clock are not measured correctly. If M.SYMBOLS is included, PERFORM also gives information on the amount of time spent in the monitor (MPASLIB).

COVERAGE ANALYSIS

The CA program provides a coverage analysis of a Lisa Pascal program. Branch counters are inserted into the source (.TEXT file) of the program under test. The output of the CA program is then compiled and linked. At the end of the program's execution, a text file is produced giving the branch numbers and the number of times each branch was executed.

To run the coverage analysis program you need to get the sources of any of the units and programs you want to analyze and the .OBJ versions of any other units that are required to link the program. You should be able to compile and link these sources without error. To add counters to a unit or program:

X(ecute CA

CA first asks for the name of the source file:

Input file -

Give it the name of the file you want analyzed. The output of CA is another source file containing the original source modified by the addition of the counters and the analysis machinery. The output file can be very large, so give it plenty of room.

Output file -

The name you give here is the name of the new source (with the branch counters added in).

The next prompt is:

Count B(ranches, P(rocedures, O(ne unit

You can count every branch (every THEN, ELSE, CASE branch, REPEAT, DO, etc), just procedure entries, or just report on the branch counters in a unit that has already been run through CA. If you ask that every branch be counted, CA also asks:

Routine to skip (<cr>> for none):

The compiler has a fixed code buffer size. If a procedure in the original program is close to the size limit, it may be impossible to add counters to every branch and still compile that procedure. The compiler error is #350, "procedure too large". If you add the counters and the compiler complains about some procedure, run CA again, and give the offending procedure name here. If more than one procedure is too large, either complain to the developers, or ask someone to make the 'routine-to-skip' code take a list of names.

The next prompt was Pete Cressman's idea:

Enable tracing?

Type 'y' and every time the program is executed you will be asked if you want to be informed about every procedure entry during execution of the program. This avalanche of names can be very tedious to watch. If you respond "n", the tracing machinery is omitted from the program and you are never asked whether it should be activated.

The next prompt is:

Data file -

The data file is the file containing the coverage analysis after the program has executed. It is a text file, so you can read it with the mouse editor. You can match each counter number up with the code it is counting by examining the output file that CA produces. At each branch you will find a procedure call of the form:

_CA_Inc(n);

where n is the counter number. All objects added by CA to the program start with the the four letters '_CA_' to try to avoid naming conflicts.

Finally, you are given a chance to place some arbitrary sequence of text in the header of the data file (date of the test, or whatever). Type (cr) to end the comments.

If you are adding counters to a unit, some of these questions are omitted because they are irrelevant.

Once CA has added the counters, you must compile the output file, generate the .OBJ file, and link it with all the units it requires. If all goes well, each time you execute the program the data file is updated. If the data file does not exist, it is created. If it does exist, the counter data it contains are added to the current counter values. The resulting data file therefore contains a record of an arbitrary number of program executions.

If you get the Linker warning:

Segment (mumble) too large

you will have to break that segment into two pieces, write a procedure to force the new segment to be resident whenever the old one was, and start over with the CA program. The problem here is that a segment can contain no more than 32 Kbytes of code or data. There is no way the CA program can tell when a segment is close to the limit. If a segment is right on the borderline, it is not inconceivable that the branch counters will cause it to overflow.

The counters pin at 32767. Programs that run in the Window Manager-OS environment are recognized, and theoretically correct code is issued, but no promises are made yet. CA increases the size of both the source .TEXT file and the final .OBJ file by about 30 percent. It may slow execution rates noticeably.

SCRIPT

SCRIPT is Colin McMaster's text formatting program. SCRIPT commands are:

Page Offset 0 in N Indent N columns from page offset Indent 0 in N Indent N columns from page offset Temporary Indent 0 in N Indent N columns from page offset It Indent N Columns from page of Indent N Columns from page offset It Indent N Columns from page of Indent N Columns from pa	Name	Default	Example	Effect
Page Number 1	ත යාත සහ ප ප සංක්ෂ දෙක සංක්ෂ අ	• & && & & &		
Page Number 1 Page Break Page Br	Page Length	66	~pl N	Define page length to be N lines
Page Break — 'Dep Start a new page Need Lines — 'ne N Make sure at least N lines remain on page Line Space 1 'Is 1 Set single or double spacing Space 1 'Sp N Space N lines Start a new line Page Offset 0 'Dep N Start a new line Page Offset 0 'In N Indent N columns from page offset Indent 0 'In N Indent N columns from page offset Indent N argin 72 'rm N Set line length to N characters Still Margin 72 'rm N Set line length to N characters Set filling mode to true Set filling mode to true Set filling mode to true Set filling mode to false Justify — 'ad Justify text to right margin Turn justification off Center 1 'Ce N Center next N lines Text 'N Display N literally Change command — 'Cc N Change SCRIPT command character to N 'L'C'R Print ritles Left, Center, Right Set number of lines above and including header Margin 1 4 'ml N Set number of lines below header Margin 2 'm2 N Set number of lines below header 'Margin 4 'm4 N Set number of lines below header 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'm4 N Set number of lines below footer 'Margin 4 'M2 N Set number of lines below footer 'M4 N Set number of lines below footer 'M5 N N N N N N N N N N N N N N N N N N N		1	'pa N	Start page numbering at N
Need Lines — ne N Make sure at least N lines reached to page Line Space 1		-	*bp	
Line Space 1	-	-	ne N	
Space 1 Space N lines Break Line		1	*1s 1	
Start a new line Page Offset 0 po N Start leftmost printing at column N Indent 1 columns from page offset Indent 0 in N Indent N columns for next line only Right Margin 72 rm N Set line length to N characters Fill	•	1	sp N	
Page Offset 0 in N Indent N columns from page offset Indent 0 in N Indent N columns from page offset Temporary Indent 0 in N Indent N columns from page offset It Indent N Columns from page of Indent N Columns from page offset It Indent N Columns from page of Indent N Columns from pa	Break Line	-	*bt	
Indent N columns from page offset Temporary Indent 0 Ti N Indent N columns for next line only Right Margin 72 Tm N Set line length to N characters Fill		0	°po N	Start leftmost printing at column N
Temporary Indent 0 Right Margin 72 Trm N Set line length to N characters Fill		0	in N	Indent N columns from page offset
Right Margin 72		- 0		Indent N columns for next line only
Fill — fi Set filling mode to true No Fill — fnf Set filling mode to false Justify — fad Justify text to right margin No Justify — faa Turn justification off Center l fex — fx 'N Center next N lines Text — fx 'N Display N literally Change command — fcc N Change SCRIPT command character to N Title — ful 'L'C'R Print titles Left, Center, Right Margin 1 4 ful N Set number of lines above and including header Margin 2 ful N Set number of lines below header Margin 3 2 ful N Set number of lines above and including footer Margin 4 4 ful N Set number of lines below footer Margin 4 6 ful N Set number of lines below footer Margin 6 ful C'R Place headers Left, Center, Right Margin 7 ful N Set number of lines below footer Margin 8 ful N Set number of lines below footer Margin 9 ful N Set number of lines below footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines below header Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margin 9 ful N Set number of lines above and including footer Margi				Set line length to N characters
No Fill — 'nf Set filling mode to false Justify — 'ad Justify text to right margin Turn justification off Center 1		_		
Justify — ad Justify text to right margin Turn justification off Center 1 ce N Center next N lines Text — tx 'N Display N literally Change command — cc N Change SCRIPT command character to N Title — tl 'L'C'R Print titles Left, Center, Right Margin 1 4 ml N Set number of lines above and including header Margin 2 2 ml N Set number of lines above and including footer Margin 3 2 ml N Set number of lines above and including footer Margin 4 4 ml N Set number of lines below header Margin 4 6 ml N Set number of lines below footer Margin 4 6 ml N Set number of lines below footer Margin 6 content of 'L'C'R Place headers Left, Center, Right Even Header — ch 'L'C'R Place even headers Left, Center, Right Codd Header — ch 'L'C'R Place odd headers Left, Center, Right Footer — cf 'L'C'R Place footers Left, Center, Right Codd Footer — cof 'L'C'R Source — so 'File' Begin printing text of File Zero Slash — nz Turn on zero slashing No Zero Slash — nz Turn of zero slashing No Zero Slash — nz Turn of zero slashing No Zero Slash — nz Turn of pascal keywords Define Macro — de VO Begin definition of macro VO Terminate Macro — am VO Append to macro VO Poletar Texas VO Poletar N Poletar Texas VO Poletar N Poletar Texas VO Poletar N Pol		-		
No Justify — na Turn justification off Center 1		(T)		
Center 1				Turn justification off
Text				
Change command — CC N Change SCRIPT command character to N Title — Ti 'L'C'R Print titles Left, Center, Right Margin 1 4 ml N Set number of lines above and including header Margin 2 m2 N Set number of lines below header Margin 3 2 m3 N Set number of lines below footer Margin 4 h Set number of lines below footer Margin 4 h Set number of lines below footer Margin 4 h Set number of lines below footer Margin 4 header — The 'L'C'R Place headers Left, Center, Right Margin 5 h L'C'R Place even headers Left, Center, Right Margin 6 h L'C'R Place even headers Left, Center, Right Margin 7 h L'C'R Place odd headers Left, Center, Right Margin 8 h L'C'R Place footers Left, Center, Right Margin 9 h L'C'R Place footers Left, C				
Title — "tl'C'R Print titles Left, Center, Right "ml N Set number of lines above and including header Margin 2 2 m3 N Set number of lines below header margin 3 2 m3 N Set number of lines above and including footer margin 4 margin 4 margin 4 margin 5 m4 m Set number of lines below footer margin 6 m4 m Set number of lines below footer margin 7 m4 m Set number of lines below footer margin 6 m4 m Set number of lines below footer margin 7 margin 8 margin 9 margin				Change SCRIPT command character to N
Margin 1 4	•			
Margin 2 2 m2 N Set number of lines below header Margin 3 2 m3 N Set number of lines above and including footer Margin 4 4 m4 N Set number of lines below footer Header — he 'L'C'R Place headers Left, Center, Right Even Header — oh 'L'C'R Place even headers Left, Center, Right Odd Header — oh 'L'C'R Place odd headers Left, Center, Right Footer — fo 'L'C'R Place footers Left, Center, Right Even Footer — of 'L'C'R etc Odd Footer — of 'L'C'R Source — so 'File' Begin printing text of File Zero Slash — rs Turn on zero slashing No Zero Slash — nz Turn off zero slashing Keywords — kw Underline Pascal keywords No Keywords — oh No Keywords — oh No Keywords Define Macro — oh No Mappend to macro VO End definition of macro Append Macro — am VO Append to macro VO Fellow Macro — oh No Marco Macro Macr				Con makes of lines shows and including header
Margin 3 2 m3 N Set number of lines above and including footer Margin 4 4 m4 N Set number of lines below footer Header — he 'L'C'R Place headers Left, Center, Right Even Header — 'C'R Place even headers Left, Center, Right Odd Header — 'C'R Place odd headers Left, Center, Right Tooter — 'Fo 'L'C'R Place footers Left, Center, Right Even Footer — 'C'R Place footers Left, Center, Right Even Footer — 'So 'File' Begin printing text of File Zero Slash — 'So 'File' Begin printing text of File Turn on zero slashing No Zero Slash — 'Nz Turn off zero slashing Keywords — 'Kw Underline Pascal keywords No Keywords — 'Nk Do not underline Pascal keywords Define Macro — 'Append to macro VO Append Macro — 'Append to macro VO Deliver macro VO Append to macro VO Append to macro VO				Cor number of lines below header
Margin 4 4 m4 N Set number of lines below footer he 'L'C'R Place headers Left, Center, Right eh 'L'C'R Place even headers Left, Center, Right oh 'L'C'R Place odd headers Left, Center, Right foo'L'C'R Place footers Left, Center, Right fo'L'C'R Place footers Left, Center, Right ef 'L'C'R etc Odd Footer footer for 'L'C'R Source footer file Zero Slash for Turn on zero slashing No Zero Slash for Turn off zero slashing No Zero Slash for Underline Pascal keywords No Keywords for Do not underline Pascal keywords Define Macro ferminate Macro fam VO Append to macro VO Polleto m				car number of lines shows and including footer
Header — The 'L'C'R Place headers Left, Center, Right Even Header — Center Center, Right Odd Header — Con 'L'C'R Place odd headers Left, Center, Right Footer — Center Center — Center —				
Even Header — 'eh' 'L'C'R Place even headers Left, Center, Right Odd Header — 'oh 'L'C'R Place odd headers Left, Center, Right Footer — 'fo 'L'C'R Place footers Left, Center, Right Even Footer — 'ef 'L'C'R etc Odd Footer — 'of 'L'C'R Source — 'so 'File' Begin printing text of File Zero Slash — 'zs — Turn on zero slashing No Zero Slash — 'nz — Turn off zero slashing Keywords — 'kw — Underline Pascal keywords No Keywords — 'nk — Do not underline Pascal keywords Define Macro — 'de VO — Begin definition of macro VO Terminate Macro — 'am VO — Append to macro VO Append Macro — 'am VO — Append to macro VO	•		m4 N	
Odd Header — 'oh 'L'C'R Place odd headers Left, Center, Right Footer — 'fo 'L'C'R Place footers Left, Center, Right Even Footer — 'ef 'L'C'R etc Odd Footer — 'of 'L'C'R Source — 'so 'File' Begin printing text of File Zero Slash — 'zs — Turn on zero slashing No Zero Slash — 'nz — Turn off zero slashing Keywords — 'kw — Underline Pascal keywords No Keywords — 'nk — Do not underline Pascal keywords Define Macro — 'de VO — Begin definition of macro VO Terminate Macro — 'am VO — Append to macro VO Append Macro — 'am VO — Append to macro VO			he LC	(Place neaders beit, benter, magne
Footer — "fo 'L'C'R Place footers Left, Center, Right ef 'L'C'R etc Odd Footer — "of 'L'C'R Source — "so 'File' Begin printing text of File Zero Slash — "zs — Turn on zero slashing No Zero Slash — "nz — Turn off zero slashing Keywords — "kw — Underline Pascal keywords No Keywords — "nk — Do not underline Pascal keywords Define Macro — "de VO Begin definition of macro VO Terminate Macro — "am VO Append to macro VO Append Macro — "am VO Append to macro VO	Even Header	***	eh LC	place even neaders beit, center, might
Even Footer 'ef 'L'C'R etc Odd Footer 'of 'L'C'R Source 'so 'File' Begin printing text of File Zero Slash 'zs Turn on zero slashing No Zero Slash 'nz Turn off zero slashing Keywords 'kw Underline Pascal keywords No Keywords 'nk Do not underline Pascal keywords Define Macro 'de VO Begin definition of macro VO Terminate Macro 'am VO Append to macro VO Append Macro 'am VO Append to macro VO	Odd Header	-	"oh 'L'C'	Place odd headers Leit, Center, Addit
Odd Footer of 'L'C'R Source so 'File' Begin printing text of File Zero Slash in Turn on zero slashing No Zero Slash inz Turn off zero slashing Keywords kw Underline Pascal keywords No Keywords ink Do not underline Pascal keywords Define Macro ink Begin definition of macro VO Terminate Macro in VO Append to macro VO Append Macro in VO Append to macro VO	Footer	00-000		
Source	Even Footer	000		
Zero Slash —	Odd Footer	∞ ∞		
No Zero Slash	Source	-		
Keywords —	Zero Slash			
No Keywords —	No Zero Slash		nz	
Define Macro — de VO Begin definition of macro VO Terminate Macro — am VO Append to macro VO Append Macro — am VO Append to macro VO	Keywords			Underline Pascal keywords
Define Macro — de VO Begin definition of macro VO Terminate Macro — am VO Append to macro VO Append Macro — am VO Polone Terminate VO		-	nk	Do not underline Pascal keywords
Append Macro am VO Append to macro VO		-		Begin definition of macro VO
Delete Total	Terminate Macro	_	^^	
a. un Dalaga magga VO	Append Macro		^am VO	Append to macro VO
HEIELE MALLY	Delete Macro		dm VO	Delete macro VO

SCRIPT OPTIONS (specified when SCRIPT is executed)

-cC Change command character to C -fFILE Send output to FILE (.TEXT is not appended for you) -k Underline Pascal keywords -1 Assume output is going to a Printronix-style printer Start page numbering at N -nN -oLIST Output only the pages given in LIST Assume printer has full control of page -p Assume output is going to a Qume-like printer -đ -5 Stop printing after each page and wait for <cr> or <ESC> -zN Set page offset to N

This version of Script does not attempt to return to its top level when it has finished with a file. Because it is trying to exit the program from a unit, it usually quits with 'fatal error l'. Do not use the options that refer to printers. To see your formatted text, use either -S or -F.

More complete documentation is available from Publications.

TERMINAL EMULATOR

Files needed: TERM.OBJ

LISA: SYSTEM.FONT LISA: TERM.MENUS LISA: SARA8F

The terminal emulator (TERM) provides a Lisa folder which is a full duplex virtual terminal. The terminal control commands implemented here are similar to those of the Hewlett-Packard 2640 and 2645, the DataMedia, Perkin-Elmer Fox and Owl, Beehive, and the DEC VT-52 terminals, as well as the "VT52" modes of the DEC VT-100 and HeathKit H19 terminals.

The terminal emulator works only with the "new" Lisa hardware. In addition, you must make a hardware modification to your Lisa: open the back of the machine and find the three large chips in the center of the visible board (the IO board). The chip nearest the power supply should already have the 10th pin from the bottom on the power supply side raised. For the terminal emulator, the 9th pin from the bottom on the same side should also be raised.

To invoke the emulator, copy the files given above, and X(ecute TERM. The emulator has three menus and a tray icon. The Speed menu sets the baud rate. Available speeds are 300, 600, 1200, 2400, 4800, 9600, and 19200 baud. 600 baud is not available on Port #1. The default speed is 300 baud.

The Port Menu determines which serial port is connected to the modem. Port #1 is the connector in the center. Port #2 is the connector nearest the power supply. The default port is #2.

The control menu has four items: Record, Play Back, Debug, and Quit. If you select Record, all characters received by the UART are saved in the file RECORD.TEXT. If you select Play Back, the contents of the file PLAYBACK.TEXT are sent to the UART just as if they had been typed. If you want to see exactly what characters are being received, including control characters and escape sequences, select Debug. To exit the terminal emulator, select Quit.

The control commands are:

Ctrl-G	Bell (screen flashes)
Ctrl-H	Backspace
Ctrl-I	Tab (8 spaces)
Ctrl-J	Linefeed
Ctrl-M	Carriage return
Ctrl-[Start Escape Sequence (see below)
Escape-@	Enter Insert Character Mode
Escape-A	Cursor Up
Escape-B	Cursor Down
Escape-C	Cursor Right
Escape-D	Cursor Left
Escape-E	Clear screen
Escape-H	Cursor home (top left corner)

Scroll down
Clear to end of screen
Clear to end of line
Insert line position
Delete line position
Delete character position
Leave Insert Character Mode
Insert character position
Absolute character positioning (Y+31, X+31)
Clear to beginning of screen
Save Cursor position
Restore saved cursor position
Erase line
Clear to beginning of line
Stand out (bold characters)
Reset Stand Out (normal characters)
Initialize terminal

All other Escape and Control sequences are ignored.

Page 100

1/ = 1 00

The rascal bevelopient bystem manual

ERROR MESSAGES

There are several error categories—I/O errors, Loader errors, trap handler errors, and Pascal Compiler errors. In most cases, you can type SPACE to return to the Monitor command line. Since nothing in the Monitor is tied to the user stack pointer, the Monitor can usually recover from errors that are fatal in the Apple II UCSD system. The Monitor's globals are hidden beneath the heap, and the Monitor code itself sits above your code space, so both are somewhat protected from inadvertent destruction.

I/O ERRORS

0	No error
1	Bad Block (Parity error)
2	Bad device number
3	Bad mode (Illegal operation)
4	Undefined hardware error
5	Lost device
6	Lost file
7	Bad file name
8	No room
9	No device
10	No file
11	Duplicate file
12	File not closed before open.
13	File not open
14	Bad format
15	Ring buffer overflow
16	Write-protect error
64	Device error

LOADER ERRORS

0	Unknown segment
1	No room in memory
2	Bad block
3	Can't read code file
4	Jump table overflow
5	SetSP at wrong place (after a physical link)
6	This loader does not handle intrinsic units
7	Too many units
8	Bad unit number
9	No INTRINSIC.LIB file
10	No unit location table
11	No segment location table
12	Cannot open intrinsic segment file
13	Cannot read file names block

14 - 1 00

FATAL ERRORS

0	Illegal index at trap handler Stack Overflow
2	Programmed Halt
3	Range value error
4	Illegal string index
5	Can't read Root Volume

PASCAL COMPILER ERRORS

Lexical Errors:

- 10 Too many digits
- ll Digit expected after '.' in real
- 12 Integer overflow
- 13 Digit expected in exponent
- 14 End of line encountered in string constant
- 15 Illegal character in input
 16 Premature end of file in source program
- 17 Extra characters encountered after end of program
- 18 End of file encountered in a comment

Syntactic Errors:

- 20 Illegal symbol
- 21 Error in simple type
- 22 Error in declaration part
- 23 Error in parameter list
- 24 Error in constant
- 25 Error in type
- 26 Error in field list
- 27 Error in factor
- 28 Error in variable
- 29 Identifier expected
- 30 Integer expected
- '(' expected ')' expected 31
- 32
- '[' expected 33
- 34
- 'l' expected
 ':' expected 35
- ';' expected 36
- '=' expected 37 ',' expected 38
- '*' expected ? 39
 - ':=' expected 40
 - 41 'program' expected 42 'of' expected

 - 43 'begin' expected 44 'end' expected

 - 45 'then' expected
 - 46 'until' expected

 - 47 'do' expected 48 'to' or 'downto' expected
- ? 49 'file' expected
- ? 50 'if' expected 51 '.' expected

 - 'implementation' expected
 - 'interface' expected
 - 54 'intrinsic' expected
 - 55 'shared' expected

Page 103

Semantic Errors:

100 Identifier declared twice

בשטוים. שובו בין שווששקשבטיסע בבובם. שוו.

- 101 Identifier not of the appropriate class
- 102 Identifier not declared
- 103 Sign not allowed
- 104 Number expected
- 105 Lower bound exceeds upper bound
- 106 Incompatible subrange types 107 Type of constant must be integer
- 108 Type must not be real
- 109 Tagfield must be scalar or subrange
- 110 Type incompatible with with tagfield type
- lll Index type must not be real
- 112 Index type must be scalar or subrange
- 113 Index type must not be 'integer' or 'longint'
- 114 Unsatified forward reference
- 115 Forward reference type identifier cannot appear in variable declaration
- 116 Forward declaration repetition of parameter list not allowed
- 117 Forward declared function repetition of result type not allowed

..

1/ = 1 00

- 118 Function result type must be scalar, subrange, or pointer
- 119 File value parameter not allowed
- 120 Missing result type in function declaration
- 121 F-format for real only
- 122 Error in type of standard function parameter
- 123 Error in type of standard procedure parameter
- 124 Number of parameters does not agree with declaration
- 125 Illegal parameter substitution
- ? 126 Result type of parameteric function function does not agree with declaration
 - 127 Expression is not of set type
 - 128 Only tests on equality allowed
 - 129 Strict inclusion not allowed
 - 130 File comparison not allowed
 - 131 Illegal type of operand(s)
 - 132 Type of operand must be Boolean
 - 133 Set element type must be scalar or subrange

 - 134 Set element types not compatible 135 Type of variable is not array or string
 - 136 Index type is not compatible with declaration 137 Type of variable is not record

 - 138 Type of variable must be file or pointer
 - 139 Illegal type of loop control variable
 - 140 Illegal type of expression
 - 141 Assignment of files not allowed
 - 142 Label type incompatible with selecting expression
 - 143 Subrange bounds must be scalar
 - 144 Type conflict of operands
 - 145 Assignment to standard function is not allowed
- ? 146 Assignment to formal function is not allowed
 - 147 No such field in this record

? 148 Type error in read

- 149 Actual parameter must be a variable
- 150 Multidefined case label
- 151 Missing corresponding variant declaration
- ? 152 Real or string tagfields not allowed
 - 153 Previous declaration was not forward
- ? 154 Substitution of standard proc/func is not allowed
 - 155 Multidefined label
 - 156 Multideclared label
 - 157 Undefined label
 - 158 Undeclared label
 - 159 Value parameter expected
 - 160 Multidefined record variant
- ? 161 File not allowed here
 - 162 Unknown compiler directive (not 'external' or 'forward')
 - 163 Variable cannot be packed field
 - 164 Set of real is not allowed
 - 165 Fields of packed records cannot be var parameters
 - 166 Case selector expression must be scalar or subrange 167 String sizes must be equal

 - 168 String too long
 - 169 Value out of range
 - 170 Address of standard procedure cannot be taken
 - 171 Assignment to function result must be done inside that function
 - 172 Loop control variable must be local
 - 190 No such unit in this file

Conditional Compilation:

- 260 New compile-time variable must be declared at global level
- 261 Undefined compile-time variable
- 262 Error in compile-time expression
- 263 Conditional compilation options nested too deeply
- 264 Unmatched ELSEC
- 265 Unmatched ENDC
- 266 Error in SETC
- 267 Unterminated conditional compilation option

Compiler Specific Limitations:

- 300 Too many nested record scopes
- 301 Set limits out of range
- 302 String limits out of range
- 303 Too many nested procedures/functions
- 304 Too many nested include/uses files
- 305 Includes not allowed in interface section
- 306 Pack and unpack are not implemented
- 307 Too many units
- 308 Set constant out of range
- 350 Procedure too large
- 351 File name in option too long

- Decent December Comment Was at

Page 105

1/ - . 00

I/O Errors:

- 400 Not enough room for code file
- 401 Error in rereading code file
- 402 Error in reopening text file
- 403 Unable to open uses file
- 404 Error in reading uses file
- 405 Error in opening include file
- 406 Eror in rereading previously read text block
- 407 Not enough room for i-code file
- 408 Error in writing code file
- 409 Error in reading i-code file
- 410 Unable to open listing file
- 420 I/O error on debug file

Code Generation Errors:

1000+ Code generator errors - should never occur

2000					_
/ [] [] []	A	A 6	I-code	547 -	 8 3

Expression too complicated, code generator ran out of registers Code generator tried to free a register that was already free

2003-2005 Error in generating address

2006-2010 Error in expressions 2011 Too many globals 2012 Too many locals

Verification Errors:

- 4000 Bad verification block format
- 4001 Source code version conflict
- 4002 Compiler version conflict
- 4003 Linker version conflict
- 4100 Version in file less than minimum version supported by program
- 4101 Version in file greater than maximum version supported by program

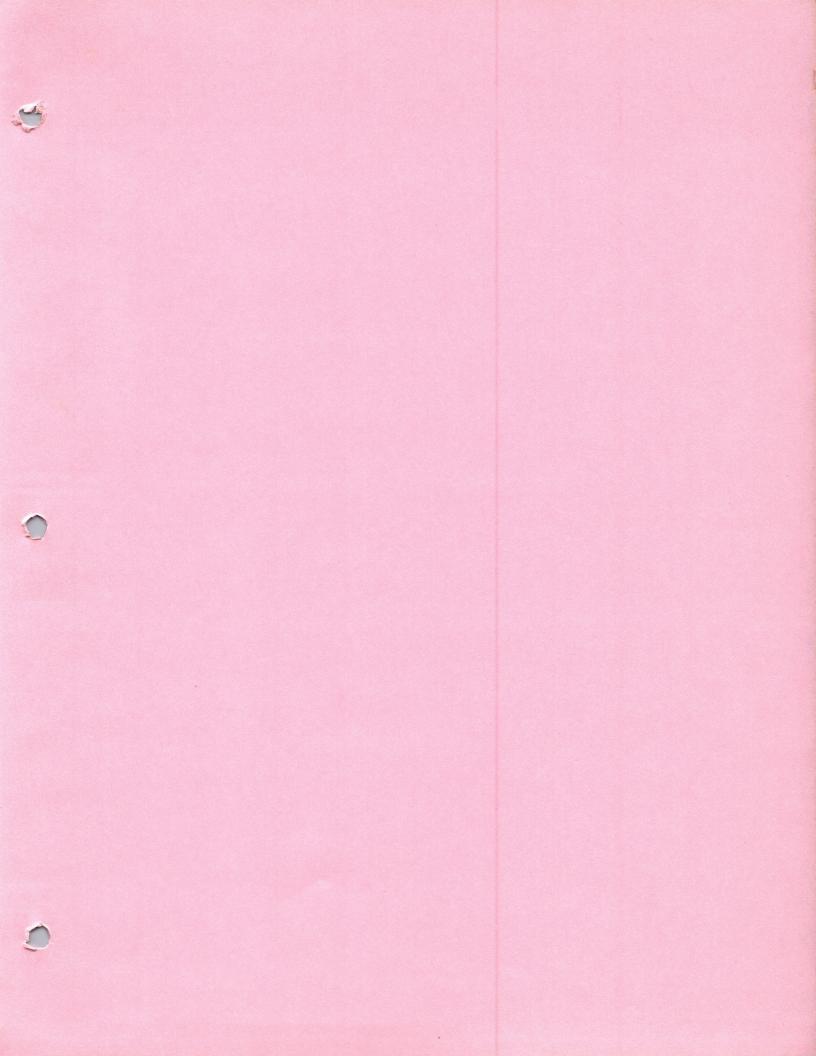
ASSEMBLER ERRORS

0. 1. undefined label operand out of range 3. must have procedure name 4. number of parameters expected 5. extra garbage on line 6. input line over 80 characters 7. not enough .IF's 8. must be declared in .ASECT before used 9. identifier previously declared 10. improper format 11. .EQU expected 12. must .EQU before use if not to a label 13. macro identifier expected 14. word addressed machine 15. backward .ORG currently not allowed 16. identifier expected 17. constant expected 18. invalid structure 19. extra special symbol 20. branch too far 21. variable not PC relative 22. illegal macro parameter index 23. not enough macro parameters 24. operand not absolute 25. illegal use of special symbols 26. ill-formed expression 27. not enough operands 28. cannot handle this relative expression 29. constant overflow 30. illegal decimal constant 31. illegal octal constant 32. illegal binary constant 33. invalid key word 34. macro stack overflow - 5 nested limit 35. include files may not be nested 36. unexpected end of input 37. this is a bad place for an .INCLUDE file 38. only labels & comments may occupy col 1 39. expected local label 40. local label stack overflow 41. string constant must be on one line 42. string constant exceeds 80 characters 43. illegal use of macro parameter 44. no local labels in .ASECT 45. expected key word 46. string expected 47. bad block, parity error (CRC) 48. bad unit number 49. bad mode, illegal operation 50. undefined hardware error 51. lost unit, unit is no longer on-line

The Process Access to the Access

```
52. lost file, file is no longer in directory
53. bad title, illegal file name
54. no room, insufficient space on disk
55. no unit, no such volume on-line
56. no file, no such file on volume
57. duplicate file
58. not closed, attempt to open an open file
59. not open, attempt to access a closed fil
60. bad format, error in reading real or int
61. nested macro definitions illegal
62. 's' or '<>' expected
63. may not EQU to undefined labels
64. must declare .ABSOLUTE before lst .PROC
65.
66.
67.
68.
69.
70. Not even a register
71. Not a Data Register.
72. Not an Address Register
73. Register Expected
74. Right Paren Expected
75. Right Paren or Comma Expected
76. Unrecognizable Operand
77. Odd location counter
78. Unimplemented Motorola directive
79. Comma Expected.
80. One operand must be a data register.
81. Dn, Dn or -(An), -(An) expected.
82. No longs allowed.
 83. First operand must be immediate.
 84. First operand must be Dn or #E
85. (An+),(An+) expected
86. Second operand must be an An
     Second operand must be a Dn
 87.
88. #<data>,Dn expected.
 89. first operand must be a Dn.
 90. An, # <displacement> expected
 91. An is not allowed with byte
 92. only alterable addressing modes allowed
 93. only data alterable addr modes allowed
 94. An is not allowed
 95. USP, SR, and CCR not allowed
 96. Cannot move from CCR
 97. Dx,d(Ay) or d(Ay),Dx expected.
 98. Only memory alterable addr modes allowed
 99. Only control addressing modes allowed
```

100. Must branch backwards to label





ROM 75: INSTALLATION DISK ERRATA

THE QUICKDRAW MOVE COMMAND

CAUSES PROBLEMS, BECAUSE MOVE '

IS ALSO A 68000 INSTRUCTION NMEMONIC.

TO USE THE QUICKDRAW MOVE', CHANGE

IT IN INTRFC: QUICKDRAW & INTRFC: QUICKDRAW

TO 'MOOV'. THEN, RECOMPILE THE

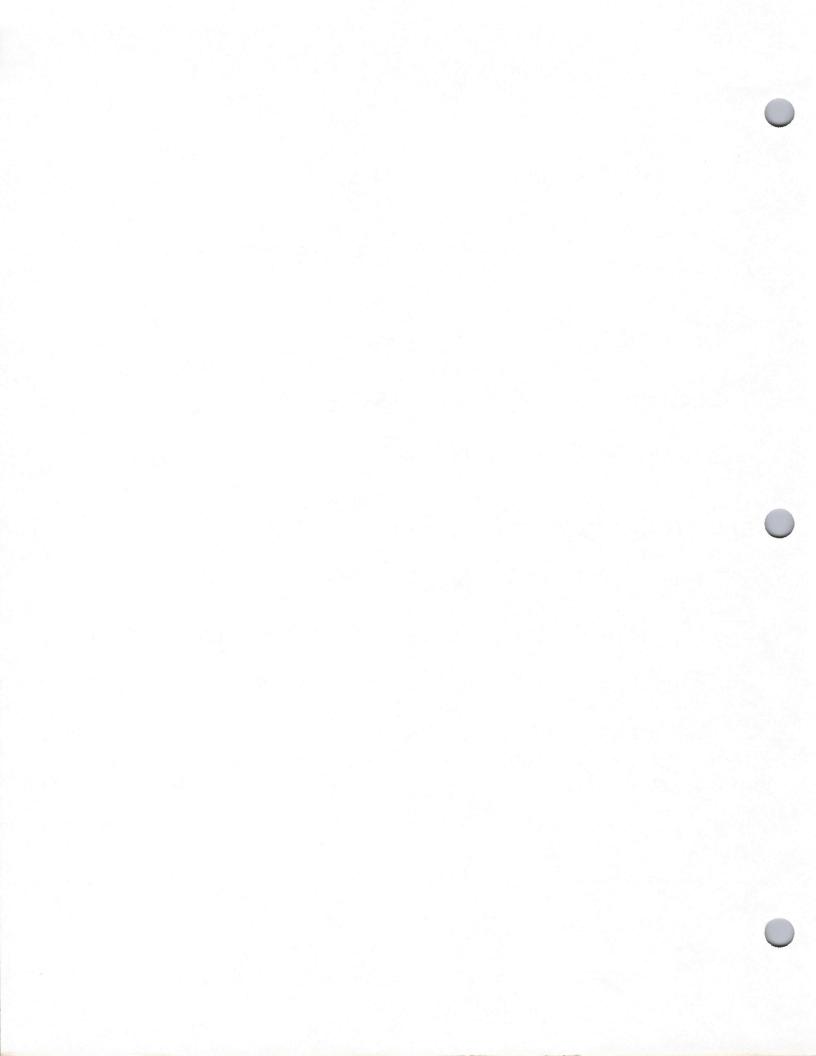
INTERFACE. USE 'MOOV' FOR NOW IN

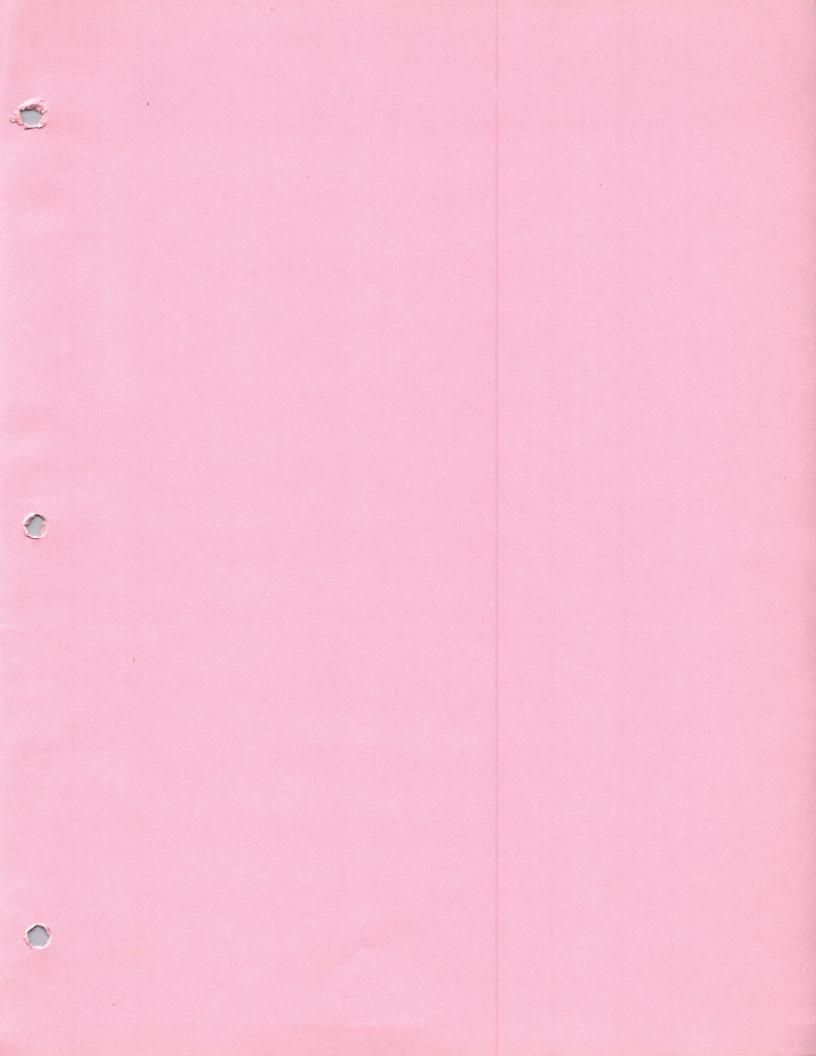
YOUR PROGRAM. LATER, A FIX WILL BE

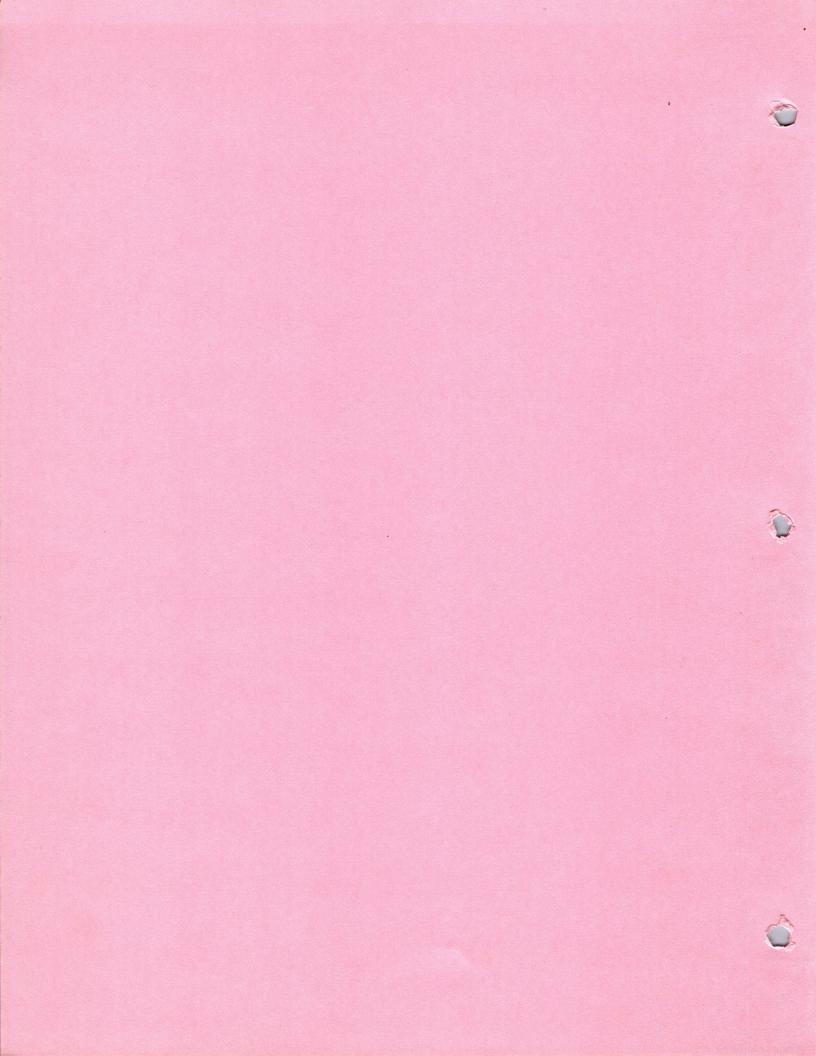
MADE TO ALLOW 'MOVE' TO BE USED FROM

YOUR PROGRAM.

NOTE: DRAW DOC GROW IS OBSOLETE,
USE DRAW GROWICON.







SOFT. GUIDES

This note details all the things to keep in mind when writing an application to make it easily localizable. It assumes a certain familiarity with the technical documentation of Macintosh. Some of the features mentioned here may not be fully documented at this time in the technical documentation.

DRAFT

SOFTWARE LOCALIZATION GUIDELINES

The time of global markets has come. Why not increase the sales of your program by selling it into international markets? These markets are developing very fast and are expected to grow at a higher rate than the US market.

To reach these markets your product needs to be fully localized. It can achieve a good penetration in international markets only if its users can understand it: having localized products is the key to success in international markets.

Macintosh unique localization technology allows you to sharply reduce the cost of localizing your product to foreign markets. Macintosh provides you with the ideal set of tools to create international products.

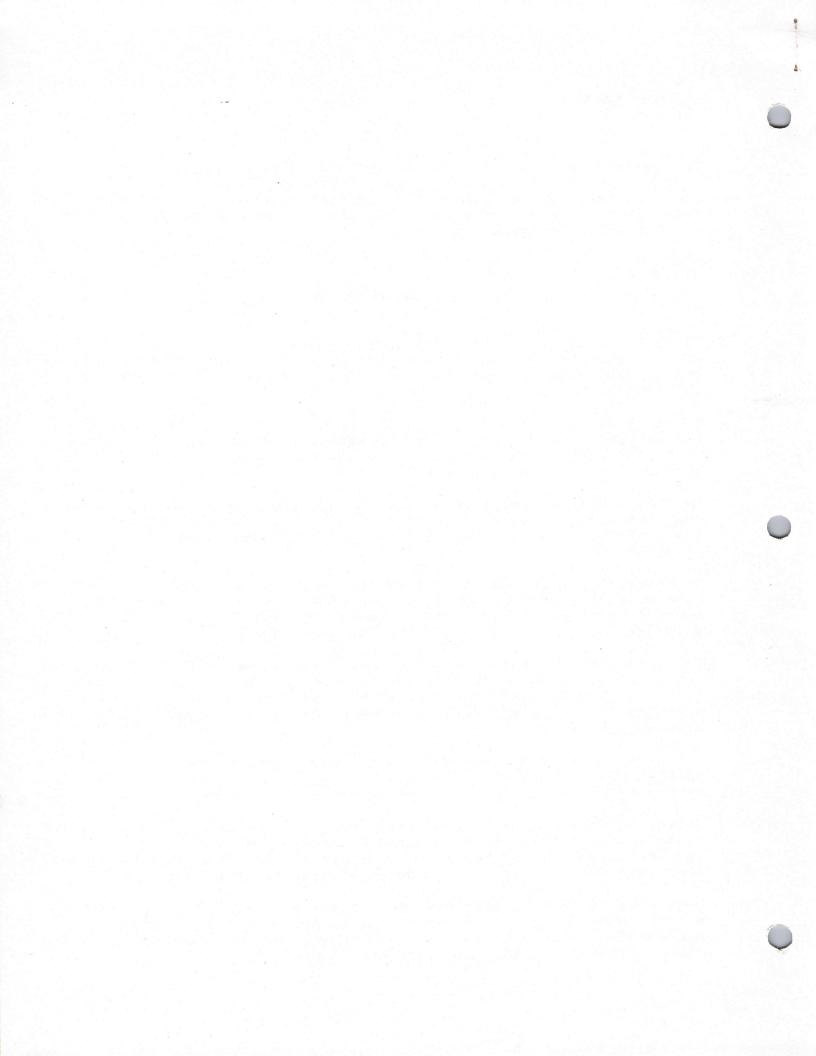
In order to get the maximum benefit from these tools, your program must be conceived from the ground up with international markets in mind. In particular, your program must make systematic use of Resources for Menus, Dialogs, Alerts, and formats.

The tools that Macintosh provides you are: predefined Resources for Menu Bars and Menu Items, Dialog Boxes, Alert Boxes, formats (number, currency, time, date), resource editor, keyboard editor, software packages for compares, time and date display, number and currency display, number input.

GENERAL DESIGN TIPS

Macintosh is a graphically oriented machine. The use of icons greatly enhances and simplifies the interaction with the user. From an international standpoint, it also simplifies the localization process. Icons are international, they don't have to be translated.

Use icons as much as possible. A good example is MacPaint where most of the commands are accessible by clicking on an Icon. Macintosh provides you with an Icon editor to create your own Icons, which can then be stored in your application resource file.



MENU ITEMS, MENU BARS, DIALOG BOXES, ALERT BOXES

Translating software can be a difficult and expensive process: the translator has to "dive" into the code to translate a program and do the layout changes required. Macintosh totally solves this problem by using resources.

TOOLS

Resources enable you to save most of the country dependent features of your program into a separate entity. This resource can then be easily modified through the resource editor. The resource editor is not only a very powerful development tool, but also the most flexible localization tool.

With the Resource Editor any non technical person can access the Dialogs, Alerts, and Menus of your program and modify them on the screen, in real time. The Resource Editor is completely graphically oriented, for example dialog zones can be grown by selecting them, clicking in their grow box, and draging the box. There are no coordinates to compute, no counting of dots. ** Will be available early 84, alpha versions for dialogs exist**

SOFTWARE DESIGN

Menu Items, Menu Bars, Dialogs, and Alert Boxes should always be put into your application resource file. This will allow a non technical person to translate, resize, and change the layout of dialog boxes, without knowing anything about the actual code of your application. Text in Dialogs, Alerts, and Menus can be edited the same way any text is edited on Macintosh. Translation simply consists in selecting the text, replacing it by the translated text, changing the layout to fit the size of the new text. Upon exiting of the resource editor, the localized program is immediately functional, no recompilation is necessary.

Your application should never rely on the length or position of strings in Menus or Dialogs. Different languages will have different word length and different word order. If your program is dependent on some string length or string order it won't work properly when translated.

LOCALIZATION

If the localization is performed by a third party, you don't have to give them your source code. This helps you keep complete control over your product. The resource editor runs on a Macintosh: no other equipment is needed for the localization phase of the program. Once a dictionary of common terms has been built, most programs can be fully translated in less than an hour by a non technical person.



CHARACTER SET AND KEYBOARDS

Foreign countries use characters that do not appear in the standard ASCII character set. This can lead to lots of problems when selling products in International markets.

TOOLS

On Macintosh, there is only one character set for all the countries which use Latin characters. This character set is common to Macintosh and Lisa. It contains all the characters needed for the major countries plus special characters and mathematical symbols (See Appendix 1 for the character set).

The character set defines the one byte codes that internally represent each character. Thus all the Macintoshes in all the countries use the same internal code for all the characters. Note that all the bits are used in the one byte code that defines a character.

Fonts contain the bit pattern that defines the shape of a specific character on the screen. Fonts are contained in a special type of resource. Some fonts may not include bit patterns for all the characters **the foreign characters may not be always there**. The system font will always have all the characters.

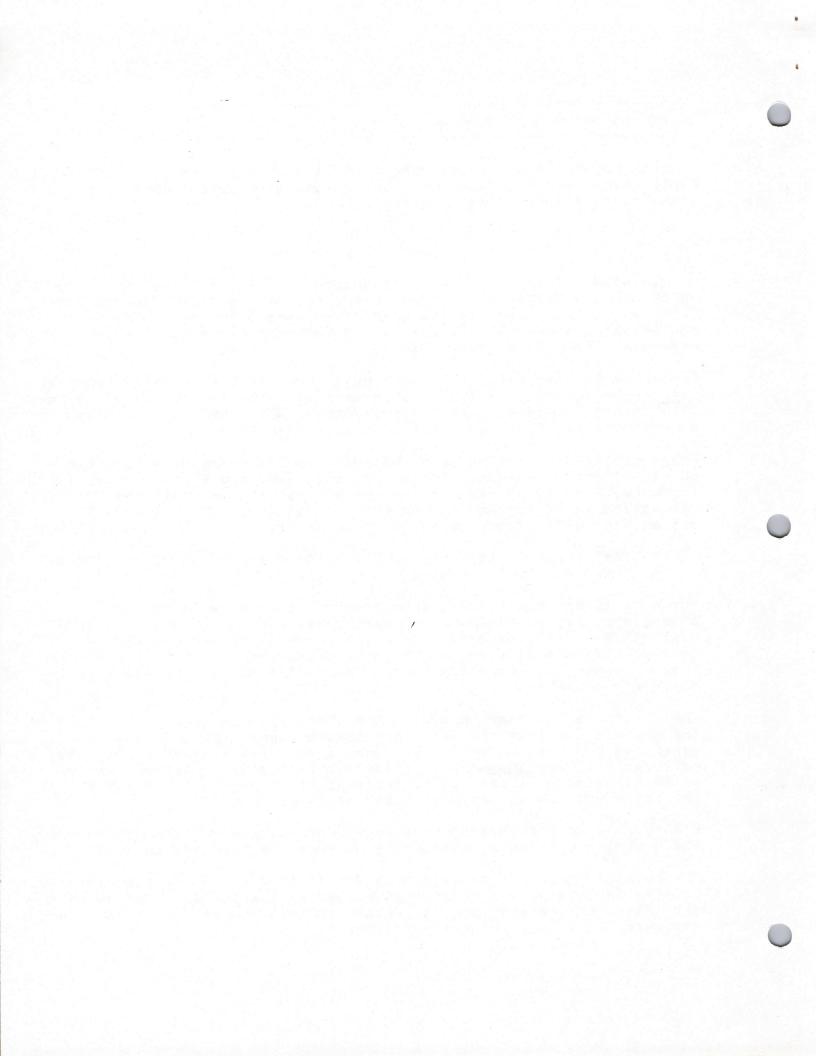
If you feel strongly that your application must have complete fonts, it can have its private fonts.

The only thing that differs from country to country is the way characters are generated. Each country has its own Keyboard (See Appendix 2 for Keyboard layouts). The whole character set can be generated from any Keyboard. The only difference is that the keys to type in order to generate a certain character may not be the same in different countries.

The option key is used to access the characters which are not shown on the keycaps. Accents are not characters by themselves. They are generated by pressing dead keys. When a dead key is pressed it does not generated any character. If the accute accent is typed, for example, nothing happens until the next character is typed. This character will be accented with an accute accent.

Please note that Keyboards have no way of identifying themselves. Only the Keyboard mapping is changed to go from one Keyboard to another.

The keyboard desk accessory allows you to look at the option keyboard. It also enables you to remap your keyboard, that is to redefine what character is generated when a certain combination of keys is pressed ** remapping is not yet implemented**.



SOFTWARE DESIGN

Do not use the 8th bit in the ASCII code to store information, it is used in our extended character set. Your program should not access the Keyboard directly. As long as it uses the toolbox routines to do so, foreign Keyboards will be transparent to your application.

Some menu commands may have keyboard equivalents in your application. Be sure to put these commands in the Menu Item definition Resource so that it can be easily edited through the Resource Editor.

LOCALIZATION

It is easy to overlook keyboard equivalents during the translation process. Be sure to explain how they have been chosen to the person who will translate your program.

ROM COMPARE

Having accented characters in the character set poses specific problems when comparing strings. There is a compare routine in ROM to handle compares.

TOOLS

The routine comes in four flavors determined by two boolean flags. The first flag is "ignore case", the second is " ignore diacritical marks".

If the "ignore case" flag is set, the comparison will be true regardless of the cases of the two characters compared. Likewise, if the "ignore diacritical" flag is set, the compare will be true regardless of the accentuation of the two characters being compared. If both flags are reset, the compare is an ASCII compare, if both flags are set it matches characters regardless of their cases and accentuation.

SOFTWARE DESIGN

Be sure to use the right kind of compare although it may not make a difference in your language. For example a simple Word processor may provide an "ignore everything" compare although there are no diacritical marks in your language.

FORMATS			



Different countries use different formats for numbers, currencies, time, date, measure units. They also have different ways of sorting lists. This can lead to lots of problems when an application is ported to another country. Macintosh gives a very elegant, yet powerful solution to this problem through the use of the resources INTLO and INTL1.

TOOLS

These resources contain information concerning number format, currency format, date format, time format, use of metric or English format, sorting. This information is stored in two predefined resource types: INTLO and INTL1.

(See Appendix 4 for the map of INTLO and INTL1).

INTL1 contains the information needed to display expanded dates and to sort. You can save space by omitting INTL1 when expanded dates and sorting are not needed.

A set of routine which allow you to add INTLO and INTL1 to your program are provided to make it easier to implement INTLO or INTL1. They may be completed by a set of access routines to allow Pascal programs to easily access the information in INTLO and INTL1 **May not be written**.

SOFTWARE DESIGN

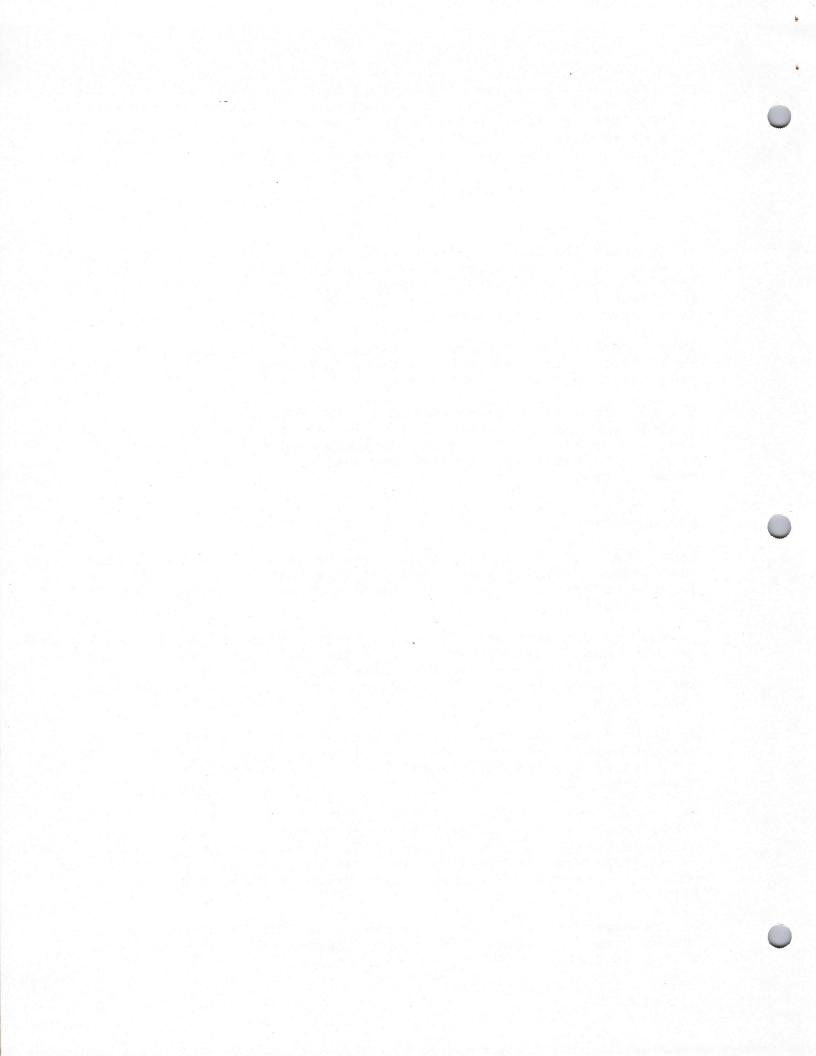
Each time your application uses anything related to these items, it must either call the appropriate routine provided in our software packages or directly look into the resources to get the necessary information.

If you use our packages, you don't need to know the detailed structure of INTLO and INTL1. We provide packages which cover: number and currency display, time and date display, magnitude compare for sorting, number input. Number and currency input and output are included in the arithmetic package **May not be written**.

Be careful to look into INTLO to get the number separator if you don't use our number input package. The number separator should also be used for list of numbers as they may appear in a function having multiple arguments.

As for any other resource, INTLO and INTL1 can live at any of these three levels: in the System Resource File, in the Application Resource File, and in the Document Resource File. Resources Files will automatically be searched for INTLO or INTL1 in the following order: Document Resource File, Application Resource File, System Resource Files.

At the developer's choice there can be an INTLO or INTLO resource in the application resource or in the document resource. There will always be a copy of INTLO and INTLO in the system resource file so



that it can be used as a default if your application does not have its private version of INTLO and INTL1.

Having your own version of INTLO or INTLO in your application's Resource File, allows your application to remember its formats independently from the disk it is moved to. If documents have their own version of INTLO or INTLO, they will keep the same format even if they are used under a foreign version of the application.

A calendar for example may not need to have its private version of INTLO or INTLO, It is sufficient for this type of application to look into the System Resource to get the necessary format information.

A program using dates may want to display dates according to the language used in its Dialogs. To achieve that it must have an INTLO in its Resource File, so that the date formats are linked to the application and not to the disk.

A Spreadsheet where numbers can be displayed as currencies would put INTLO with each document. Data integrity has to be preserved: it is not acceptable to have amounts labeled in Dollars suddenly displayed as Francs because a different version of the application is used. There must be an INTLO in each document resource file. Upon creation of the document, the default formats can be read from the application INTLO.

If the structure of your documents depends on the sorting sequence, INTL1 must be included in the document resource file. An example of that is a binary tree where part of the structure information is in the file itself and the rest in the magnitude compare.

LOCALIZATION

INTLO and INTL1 (except for sorting) are easily modified through the resource editor. This allows user to customize their formats, provided that your program makes systematic use of these resources.

It allows you to produce masters for each country that have the proper formats on them (See Appendix 3 for the most common formats in each country).

Of course if you feel that users of your program must be able to modify these formats "on the fly", you can include a routine that directly modifies INTLO or INTL1 from within your program. An example of that would be for a wordprocessor to offer the capability to switch from English to Metric rulers from within the application.



The Character Set

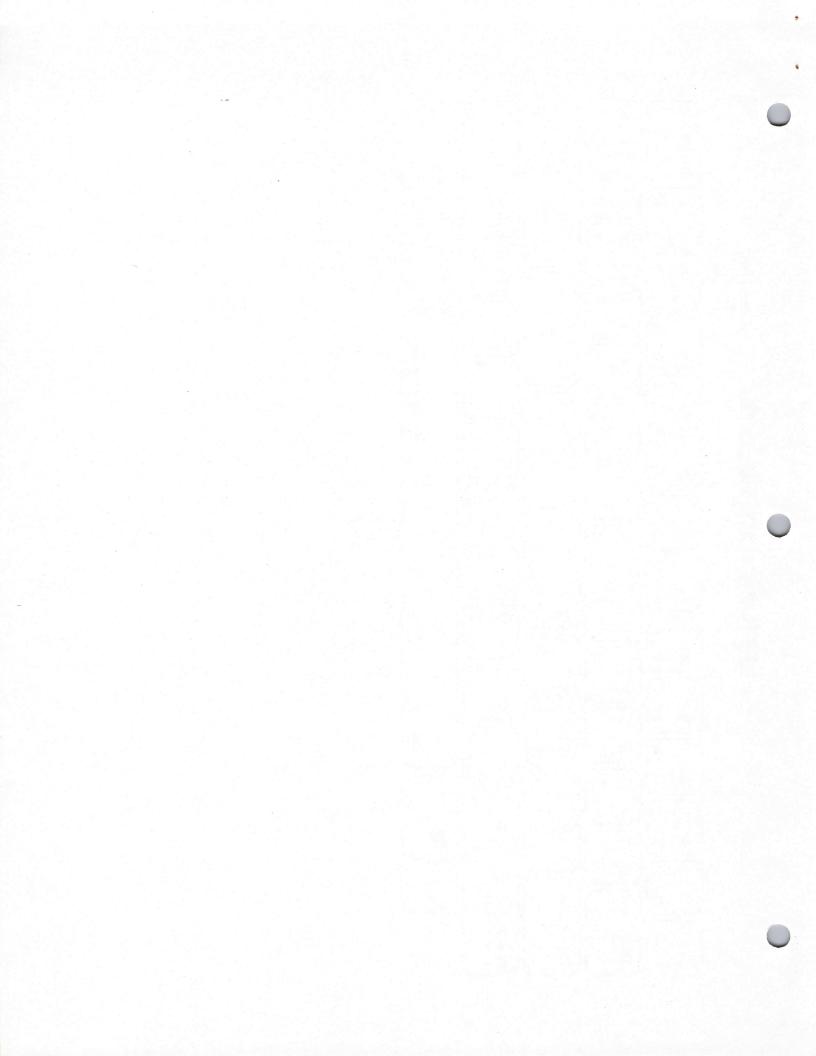
0	1	2	3	4	5	6	7	8	9	A	В	С	D	E	F
MAL.	DLE	27	0	@	Р	•	p	Ä	ê	+	∞	i	-		
SOH	DC1	!	1	A	Q	a	q	Å	ë	•	土	i	_		
STX	DC2	•	2	В	R	b	r	Ç	Í	¢	۷	_	u		
ETX.	003	#	3	С	S	С	S	É	ì	£	2	1	***		
EOT	DC4	\$	4	D	T	d	t	Ž	Î	§	¥	f	4		
ENQ	MAK	%	5	E	U	е	u	Ö	ï	•	μ	~	y		
ACK	SYM	&	6	F	٧	f	٧	Ü	ñ	91	6	Δ	÷		L
BEL	ETB	•	7	G	W	g	W	á	Ó	В	Σ	«	♦		
BS	CAN	(8	H	X	h	X	à	ò	®	π	»	ÿ		L
нт	EN .)	9	1	Y	i	У	â	ô	©	π	•••			
LF	SUB	*	:	J	Z	j	Z	ä	Ö	-	S				
VT	ESC	+	;	K]	k	{	ã	õ	•	<u>a</u>	Á			
FF	FS	,	<	L	1	I	1	å	ú		Q	Ã			
CR	20	1-	=	M	1	m	}	ç	ù	+	Ω	Õ	1		
50	RS		>	N	•	n	~	é	û	Æ	æ	Œ			1
SI	US	1	?	0		0	DEL	è	ü	Ø	Ø	œ	1		
HON-	-PRINTI	NG NG		scii—							NON-	-ascii			

Symbol for a non-breaking space. Prints a blank character, same width as numbers.

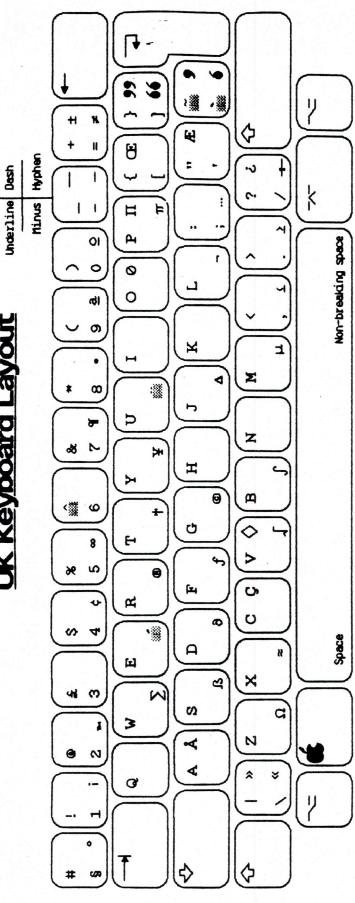
Lisa 1.0 Character Set 8 NOV 82
Revision 2 1 SEP 83

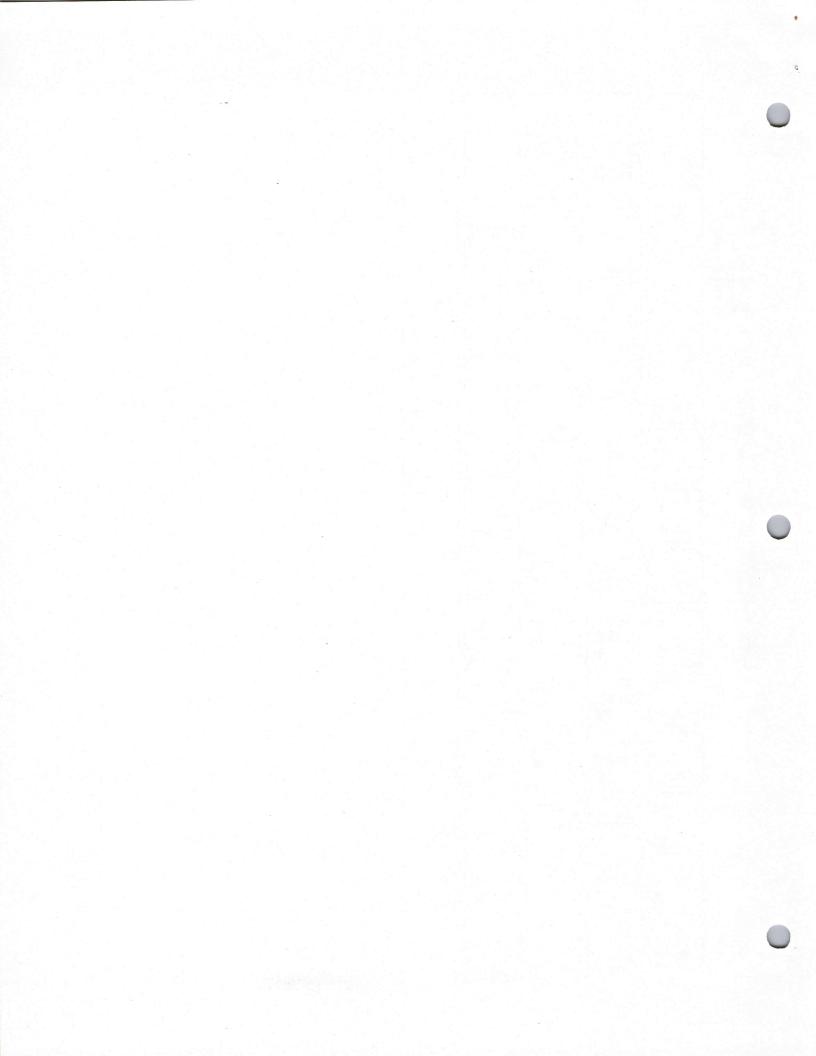


Underline Dash Minus Hyphen		Enter Option
S Keyboard Layout	2	Non-breaking space
3	Tab Tab Caps Lock A A S B B Caps Lock Option space	

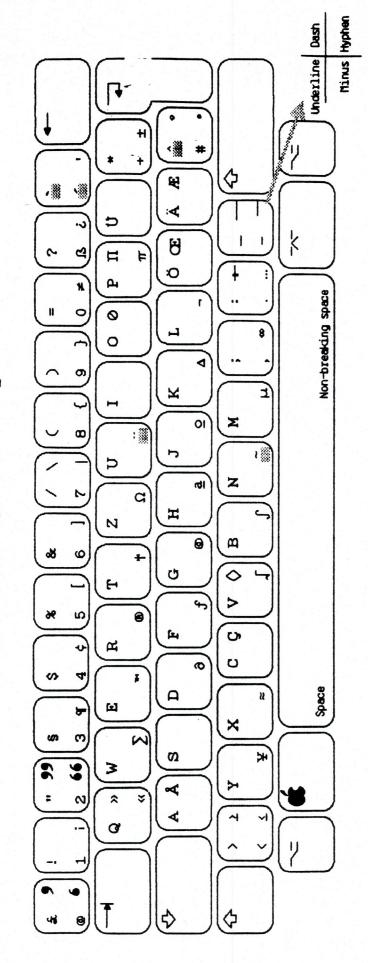


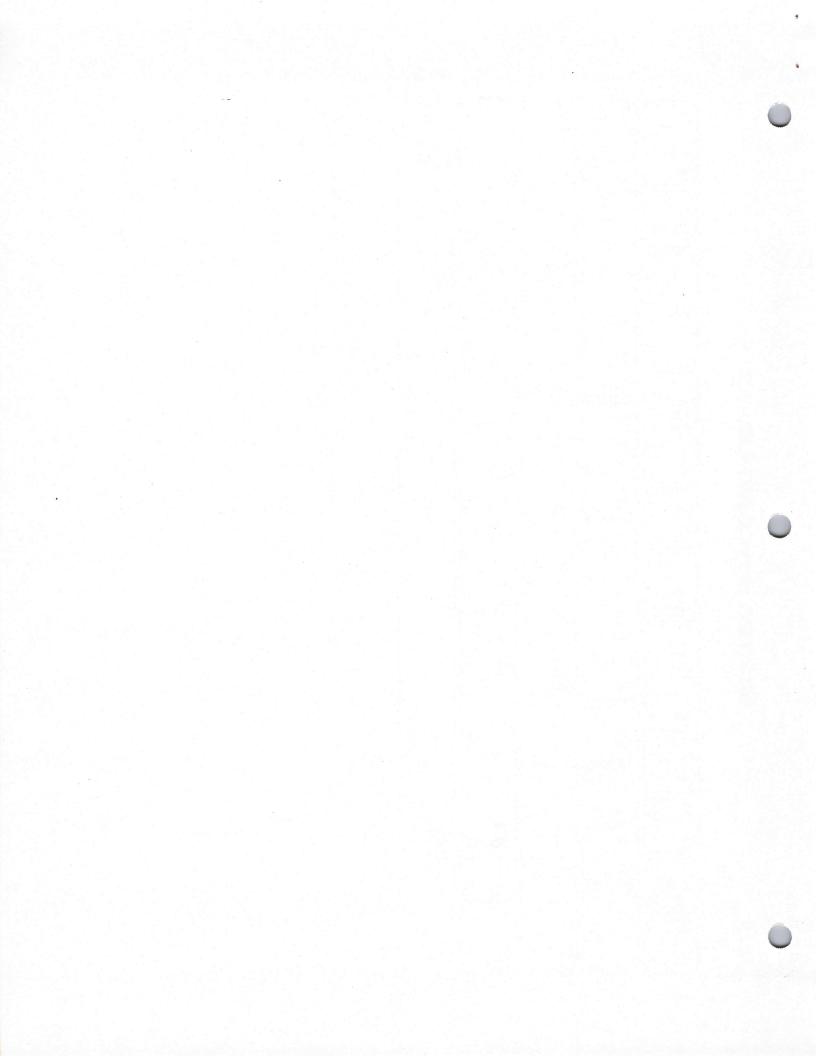
UK Keyboard Layout



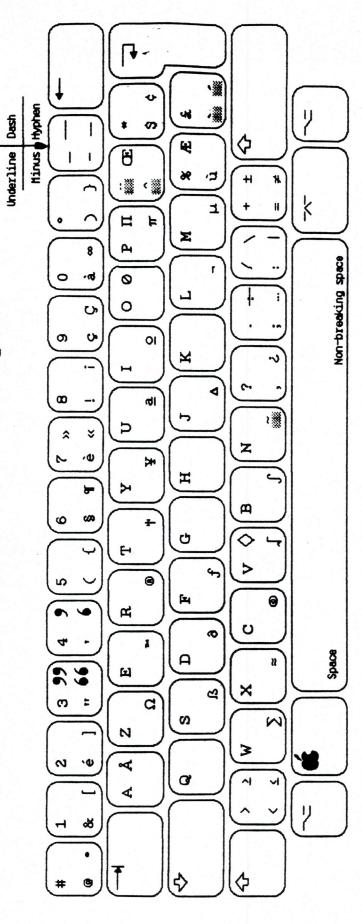


German Keyboard Layout

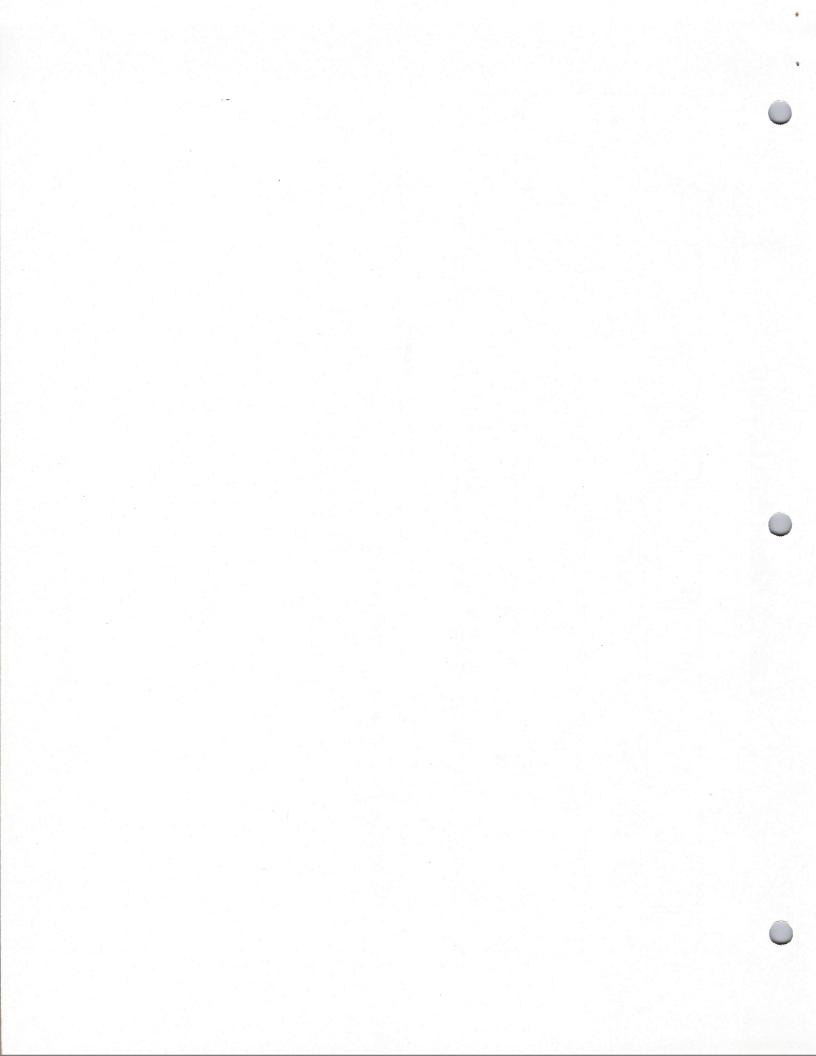




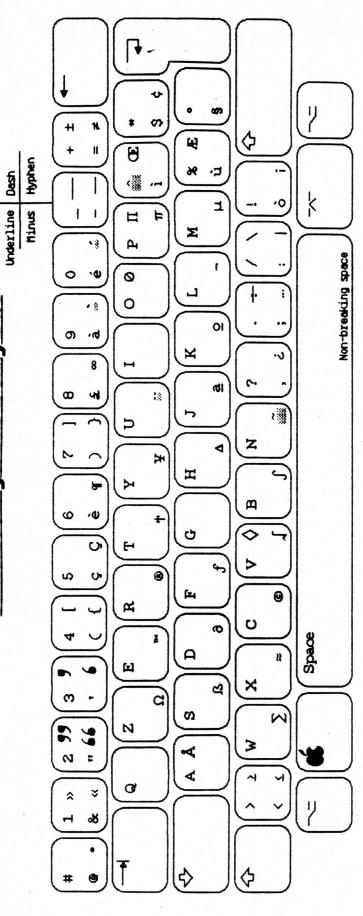
French Key Layout



ATTENTION: Caps look affects number keys tool

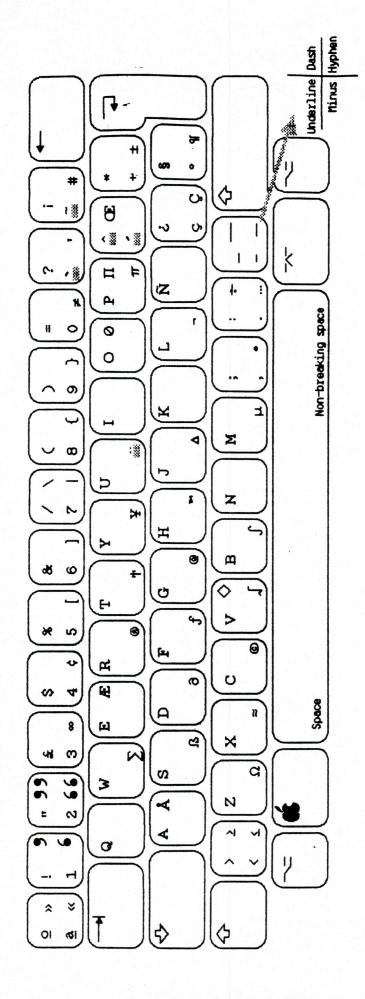


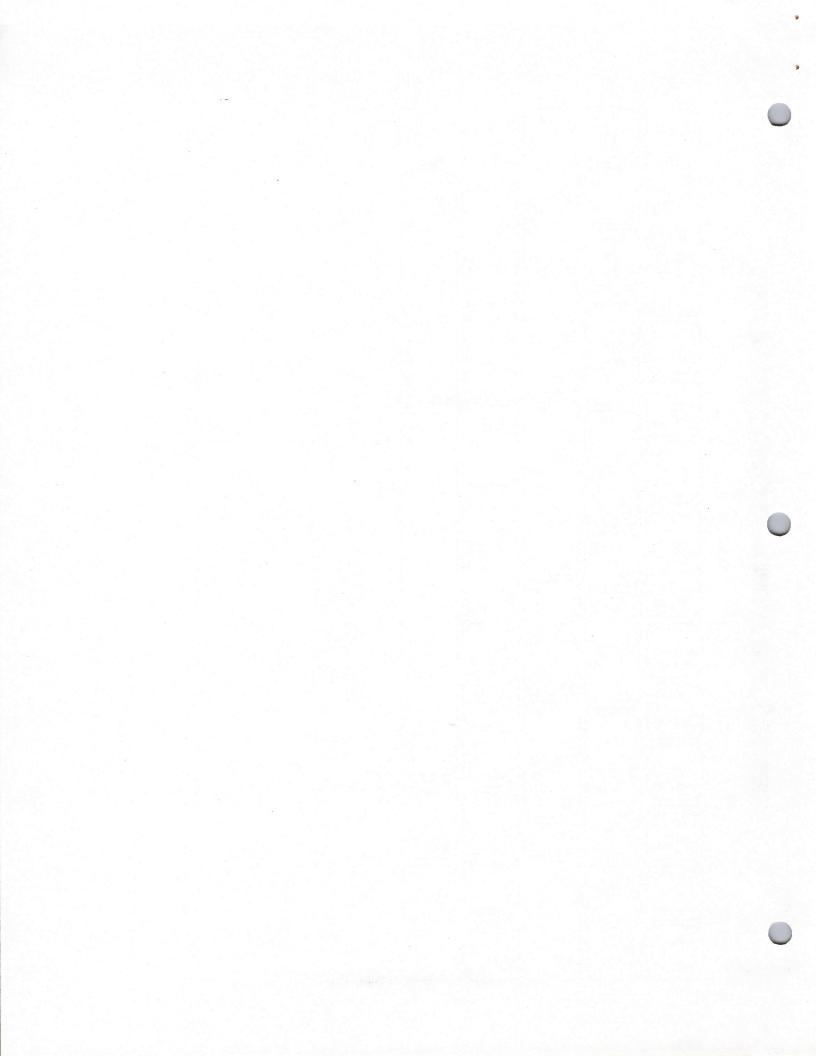
Italian Keytward Layout



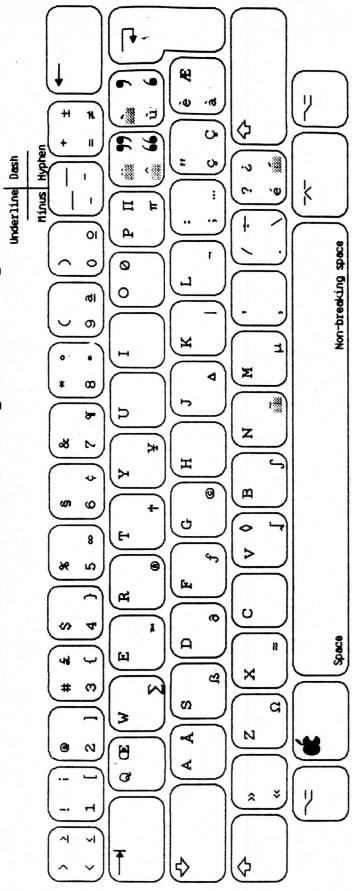


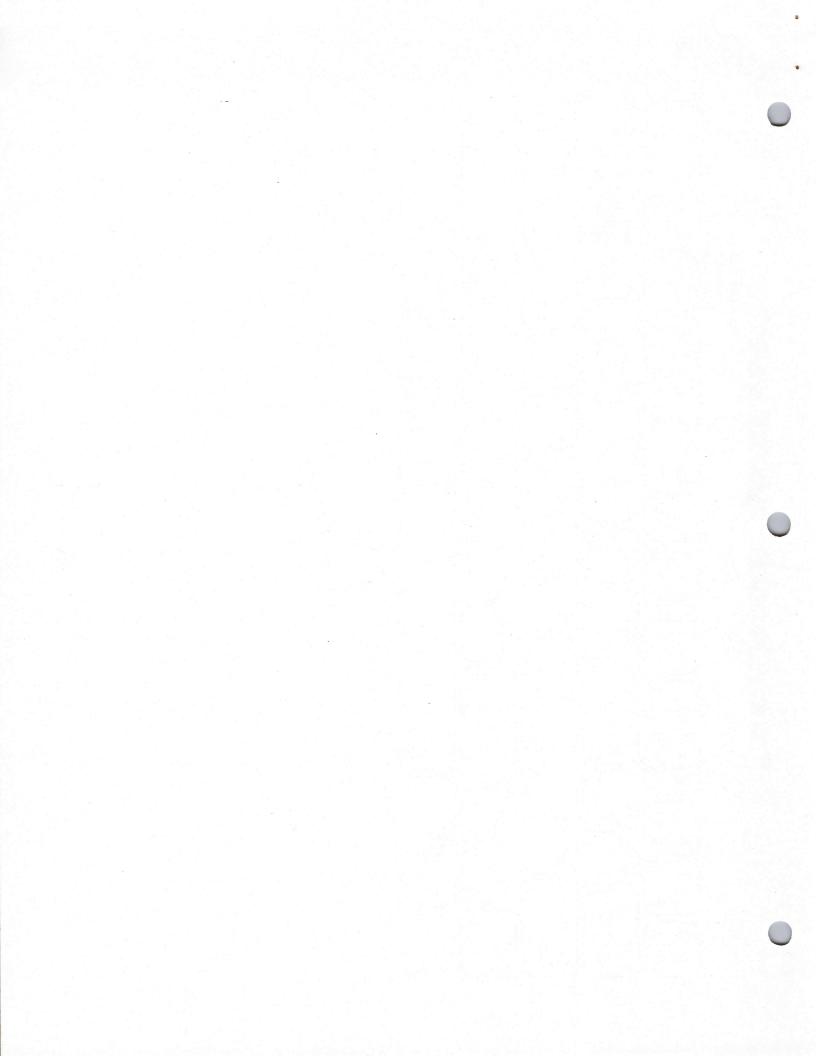
Spanish/Latin American Keyboard Layout





French/Canadian reyboard Layout





APPENDIX 3 Dates Formats

			Date	Expanded date
us	mm/dd/yy	3/31/83	6/3/83	Thursday March 31, 1983
UK	dd/mm/yy	31/03/8	33 03/06/83	Thursday March 31, 1983
Germany	dd.mm.yy	31.03.8	33 3.06.83	Donnerstag den 31. Marz 1983
France	dd/mm/yy	31/03/8	3/06/83	Jeudi 31 mars 1983
aly	dd/mm/aa	31/03/8	03/06/83	giovedì 31 Maggio 1983
	US/UK	Germany	France	Italy
Days	Monday Tuesday Wednesday Thursday Friday Saturday Sunday	Montag Dienstag Mittwoch Donnersdag Freitag Sonnabend Sonntag	Lundi Mardi Mercredi Jeudi Vendredi Samedi Dimanche	lunedi martedi mercoledi giovedi venerdi sabato domenica
Month	January February March April May June July	Januar Februar Marz April Mai Juni Juli	janvier fevrier mars avril mai juin juillet	Gennaio Febbraio Marzo Aprile Maggio Giugno Luglio
	August September October	August September Oktober	aout septembre octobre	Agosto Settembre Ottobre

novembre

decembre

Novembre

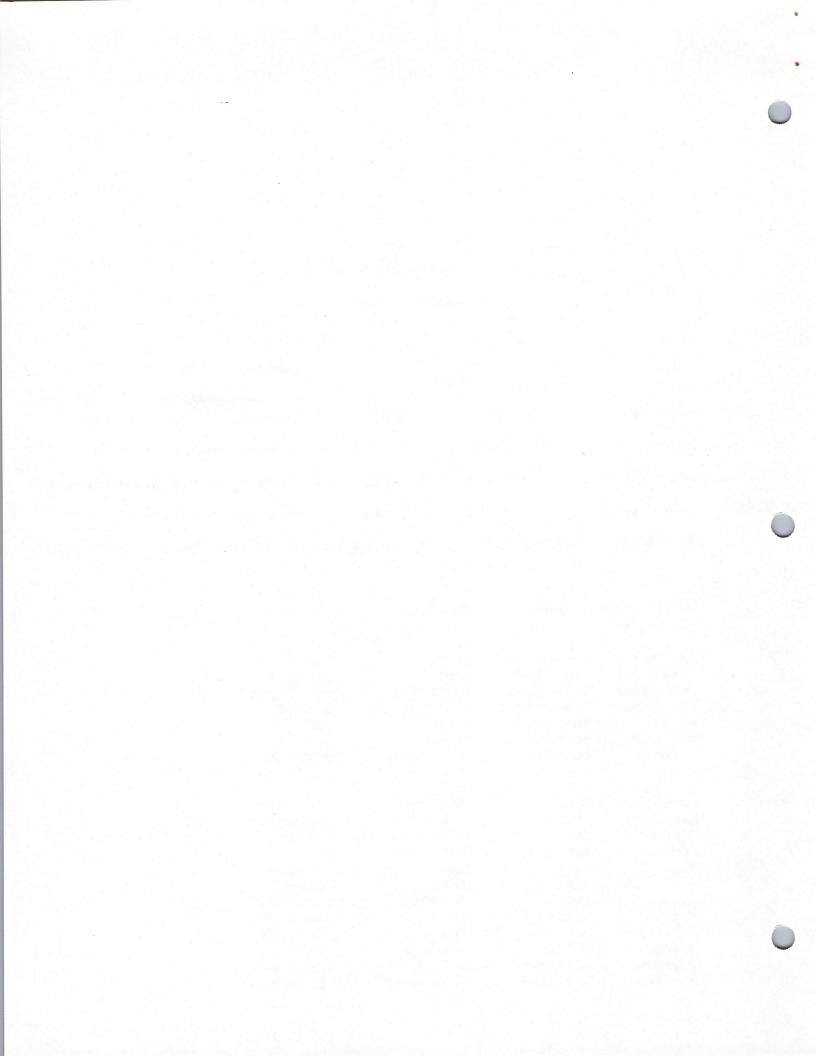
Dicembre

November

December

November

Dezember



APPENDIX 3
Time Formats

	US	UK	Germany	France	Italy		
Time	11:30 am	11:30	11.30 Uhr	11:30	11,30		
	9:05 am	09:05	9.05 Uhr	9:05	9,05		
	11:20 pm	23:20	23.20 Uhr	23:20	23.20		



APPENDIX 3

Number formats

US	UK	Germany/Italy	France
Number 1,234,567.89	1,234,567.89	1.234.567,89	1 234 567,89
Decimal 0.9876 number	0.9876	0,9876	0,9876
Separator, (assumes that numbers are entered without separators other than the decimal point)	· ,	;	;

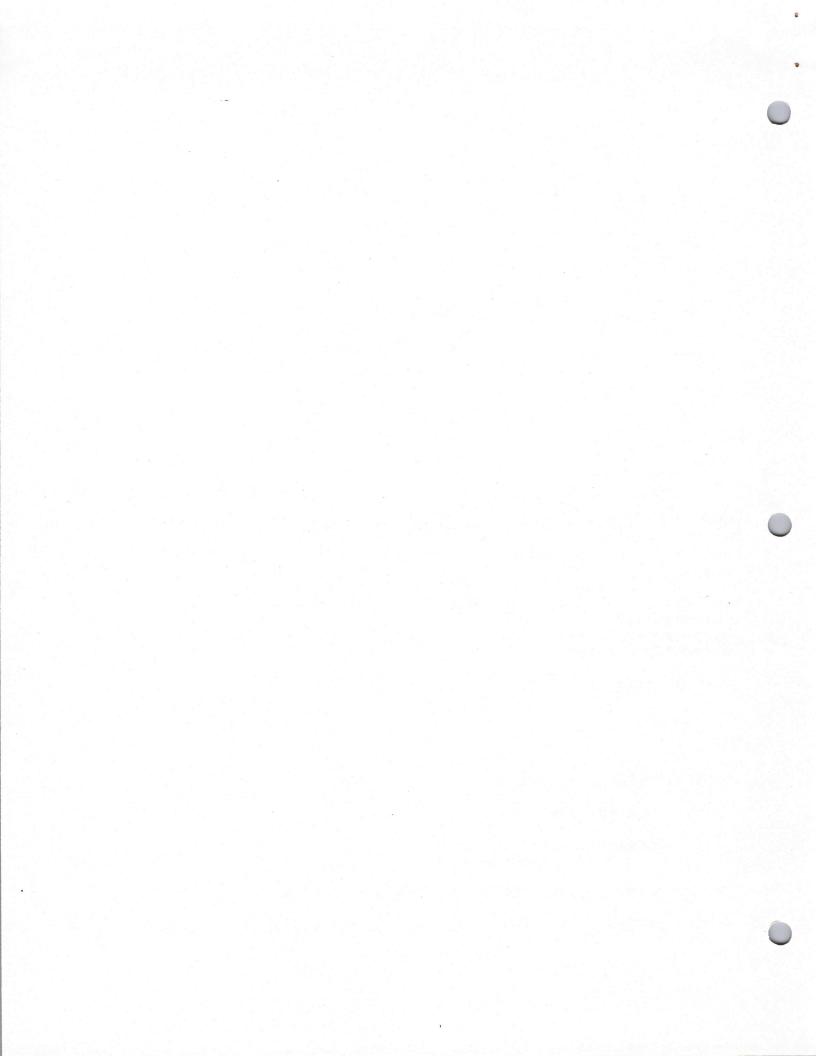
Example: Number1=4132.2

Number 2=3.14159

Separator= "," (US)
";" (Germany)

4123.2 , 3.14159 entered from the Keyboard for US

4123,2; 3,14159 entered from the Keyboard for Germany

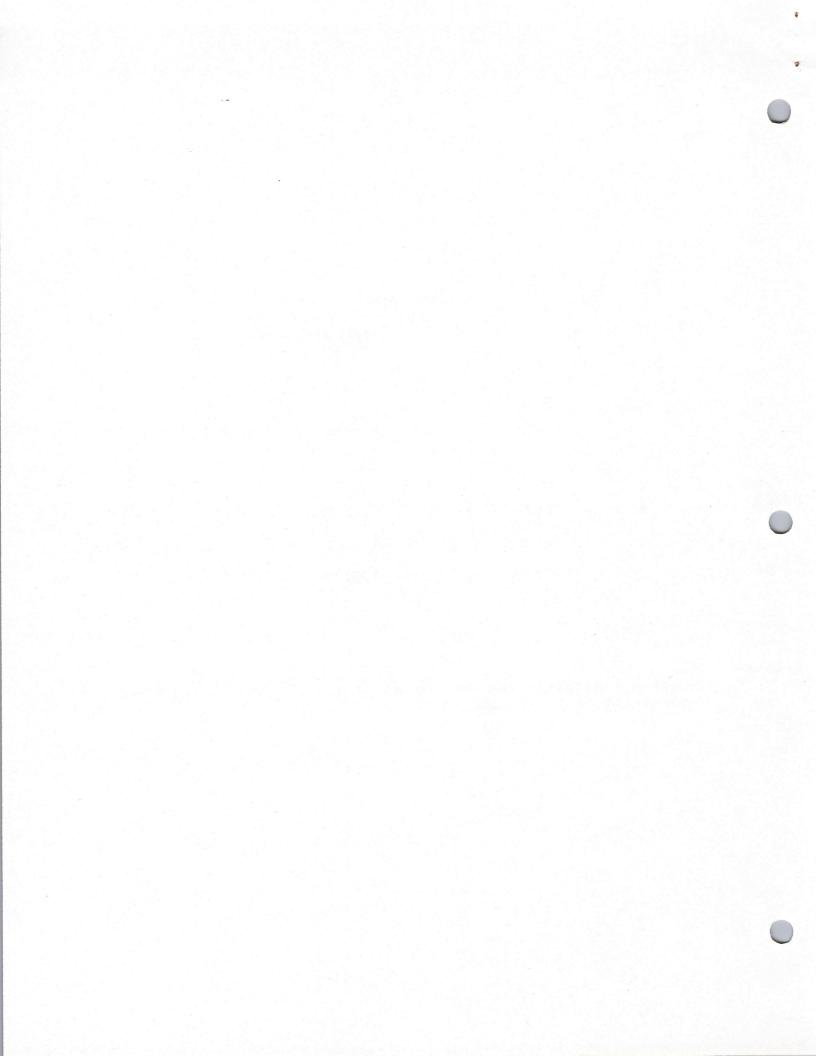


APPENDIX 3
Currency representation

	US	UK	Germany	France	Italy
Currency symbol	\$0.23	£ 0.23	0,23 DM	0,23 F	L. 0,23
egative	(\$0.23)	(£0.23)	- 0,23 DM	- 0,23 F	LIT0,23
Without decimal	\$345.00	\$ 345	325,00 DM	325 F	L. 345

Note:

Thousand separators and decimal point are the same in currency representation as in numbers $% \left(1\right) =\left\{ 1\right\}



f.TEXT

File

International Date and Time (Etc) Routines

written by Andy Hertzfeld 23-Oct-83

These routines help maintain an international parameter block as a resource of type "INTL". They translate a long integer "seconds" count into a string whose format is specified by the parameter block.

Modification History:

03-Nov-83 АЈН Added "wantSeconds" parameter to time formatting

INCLUDE ROMTOOI:SYSEQU.TEXT
INCLUDE ROMTOOI:GRAFEQU.TEXT
INCLUDE ROMTOOI:SYSMACS.TEXT
INCLUDE ROMTOOI:TOOIGUT
INCLUDE ROMTOOI:TOOIGUT
INCLUDE ROMTOOI:TOOIMACS.TEXT
INCLUDE ROMTOOI:TOOIMACS.TEXT
INCLUDE ROMTOOI:TOOIMACS.TEXT

;INTL resource type definition \$494E544C . EQU IntlRType

: Local Variable Equates

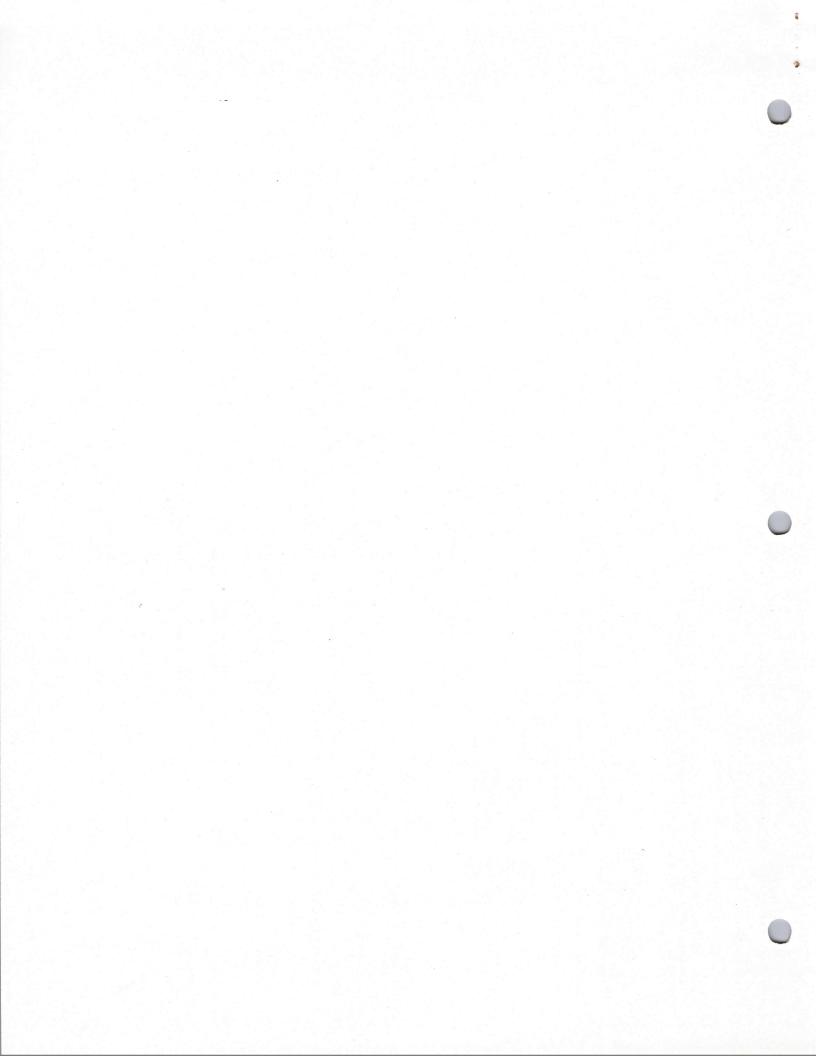
YearInteger .EQU -16
MonthInteger .EQU -14
DateInteger .EQU -12
HourInteger .EQU -8
SecondInteger .EQU -6
DayInteger .EQU -6

; International Parameter Block Definitions

; Parameter Block 0 Offsets

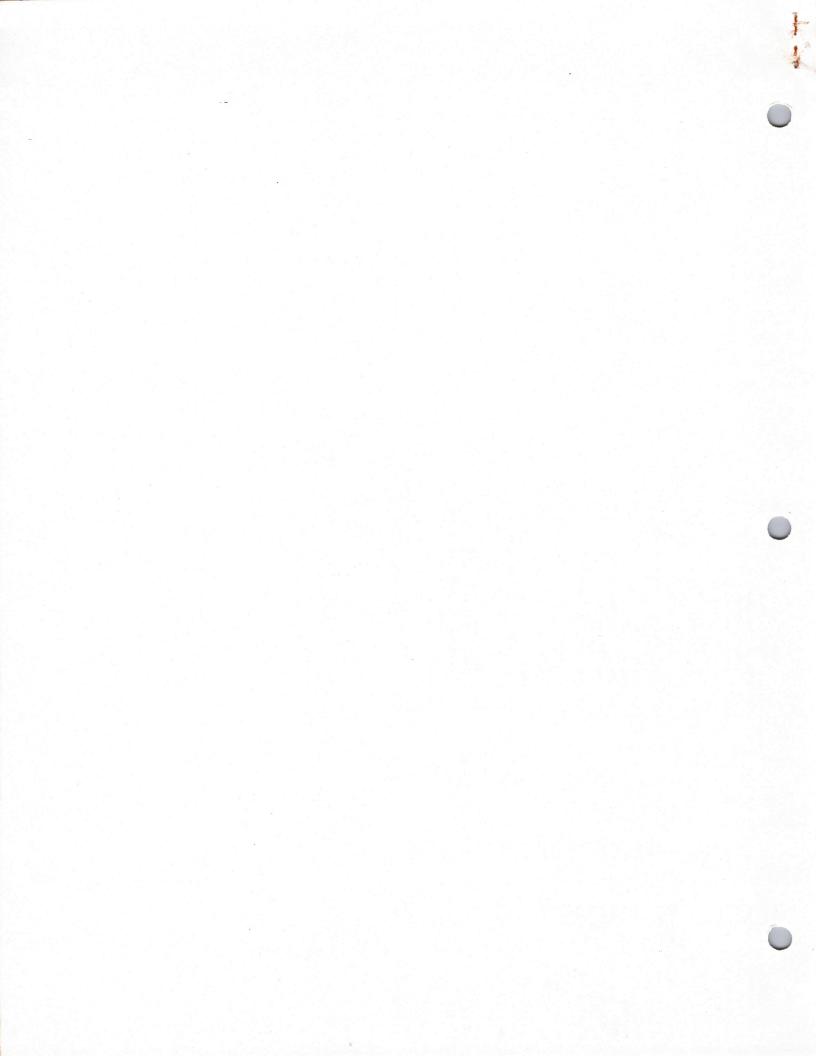
thousands separator character; seperator character for dates ; leading zero flags for dates ;12 or 24 hour time mode flag ;boolean for English/Metric ;order for short form dates suffix string for AM times suffix string for PM times time separator character; ; list separator character time leading zero flags; decimal point character; ;3 byte currency symbol currency format flags suffix for 24 hr mode; version word 10 11 11 16 20 20 21 0-0 6 9 N 8 6 E C C . EQU . EQU . EQU . EQU . EQU . EQU EQU. IntiDateOrder IntiSDLZFlags IntlTimeMode IntlOversion IntiDecPoint IntiTLZFlags IntlAMSuffix IntiPMSuffix Intl24Suffix IntiThouSep IntlListSep IntlCurSym IntlCurFmt IntlMetric IntibSep IntiTSep

; Parameter Block 1 Offsets

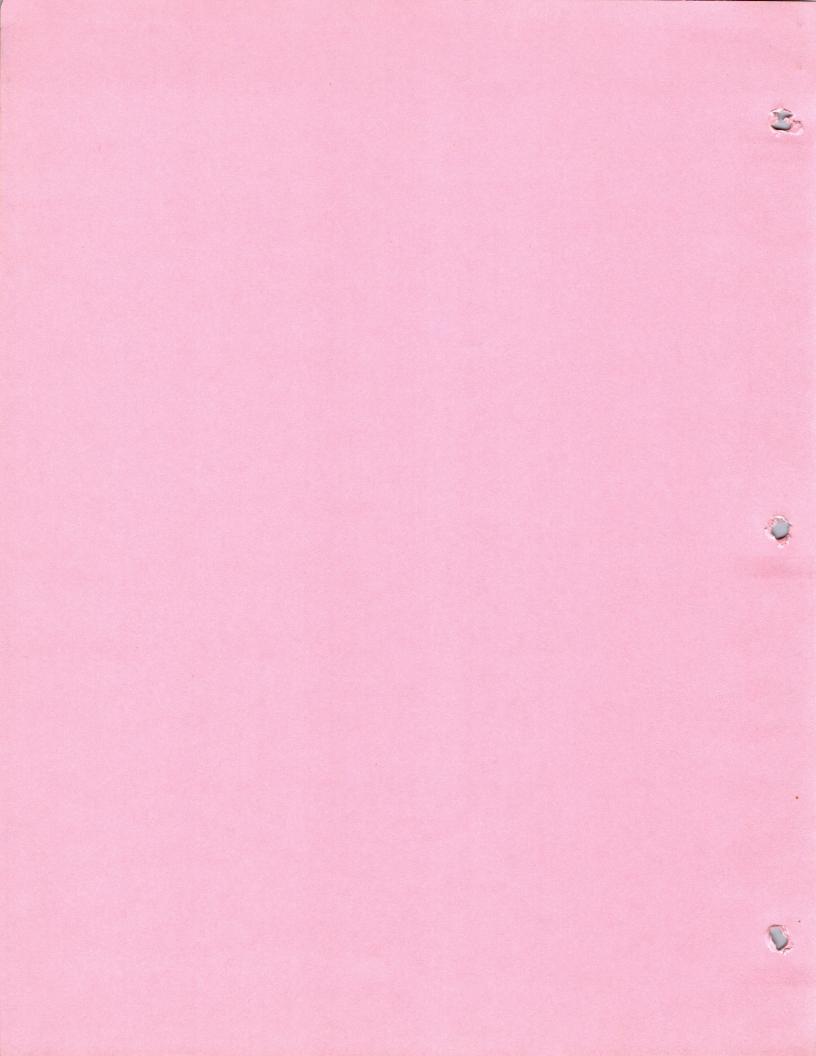


string for Sunday	string for Monday	string for Tuesday	string for Wednesday	string for Thursday	;string for Friday	string for Saturday	string for January	string for February	string for March	string for April	string for May	string for June	string for July	string for August	for	string for October	; string for November	string for December	; boolean for supressing day of week	; boolean for selecting order	;boolean for day# leading zeros	; long date prefix string	; long date day seperator string	; long date month seperator string	; long date date seperator string	; long date year suffix string	; version word (after dont care byte)	string magnitude compare tables
	•																											
0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	272	288	304	305	306	307	311	315	319	323	328	330
. EQU	. EQU	. Equ	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. Equ	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. EQU	. Equ	. EQU	. EQU
IntlSunday	IntlMonday	IntiTuesday	IntiWednesday	IntiThursday	IntiFriday	IntlSaturday	IntlJanuary	IntlFebruary	IntiMarch	IntlApril	IntlMay	Intloune	IntlJuly	IntlAugust	IntlSeptember	IntlOctober	IntlNovember	IntiDecember	IntiDaySupress	Intllong0rder	IntlLDayLZFlag	IntlStr1	Int Str2	Int Str3	Int 1 Str 4	IntlStr5	IntllVersion	IntiMagTable

i







FP68K DOCS

User's Guide -- an overview of FP68K ELEMS68K and their design philosophy.

Programmer's Guide -- hints on how to build the packages, and how to modify them, if necessary; details about system dependencies involving the state area. Includes register map templates.

System Interface -- how FP68K and ELEMS68K affect their execution environment.

High-Level Interface -- the SANE Pascal unit and assembly macros.

Integer Conversion Tests

Binary-Decimal Conversion Tests

IEEE Tests -- a set of test vectors designed for this style of arithmetic and distributed through the standards subcommittee

Binary-Decimal Conversions -- what is available through the SANE interface, and what FP68K provides at the low level.

A sample parser and formatter from the SANE interface is shown.

P754 stuff -- papers related to the arithmetic standard.

FPxxx.TEXT -- source files for FP68K, except for binary-decimal conversions

FBxxx.TEXT -- source files for binary-decimal part of FP68K

SAXXX.TEXT -- SANE68.TEXT -- SANE interface section
SAIMP68.TEXT -- SANE implementation section
SAASM68.TEXT -- SANE assembly procedures
SAMAC68.TEXT -- EQU's and MACRO's for assembly interface

DOxxx.TEXT -- documentation using SCRIPT formatter, with macros in DODRIVER.TEXT

TVxxx.TEXT -- IEEE test vector files, required operations

TWxxx.TEXT -- IEEE test vector files, appendix funtions

TDxxx.TEXT -- Test vector driver program files

ITxxx.TEXT -- integer <--> extended conversion tests

IOxxx.TEXT -- binary <--> decimal conversions tests

Zyyyy.OBJ -- executable test programs

ELxxx.TEXT -- elementary transcendental and financial functions

Introduction

The 68000 software floating-point packages, FP68K and ELEMS68K, appear much like simple subroutines but their interaction with the host system is somewhat more subtle. This document indicates possible trouble spots. It is intended for system implementors, rather than users of FP68K and ELEMS68K.

The following sections describe the various issues in turn.

Registers and stack used

FP68K and ELEMS68K receive all of their parameters on the stack. They save and restore all of the CPU registers across calls, except that DO is modified by the REMAINDER operation. FP68K modifies the CPU Condition Code Register as described later.

As detailed in the "Program Notes" document, FP68K typically uses up to 41 words of stack beyond the input parameters. The only exceptions are the binary-decimal conversion and nextafter routines, which may use up to 120 words beyond the input parameters. ELEMS68K uses at most 30 words of stack for temporary storage.

Single entry point

FP68K has just one entry point -- with the label 'FP68K'. When invoked, FP68K expects the return address on the stack, followed by a one-word opcode described in the user's guide. Beyond the opcode are up to three operand addresses (depending on the operation). Note that because the operands are passed by address, they must be in memory, NOT IN THE REGISTER FILE.

If FP68K is to be invoked by a mechanism like the A-line trap, care must be taken that stack is set up properly. Depending on the system, it should be possible to execute FP68K either as a subprogram linked to an application program, or as system-provided utility.

Because of the varying number of input parameters, it is impossible to call FP68K directly from Pascal, since the number of parameters is fixed when the EXTERNAL procedure is defined. In any case programmers should use the provided Pascal interface, called SANE (Standard Apple Numeric Environment).

 ${\tt ELEMS68K}$ has a similar design, but is configured as a separate package for modularity.

Exit points

Typically, FP68K exits by clearing all input operands from the stack and jumping to the return address.

However, a 'halting' mechanism is provided whereby control is transferred from FP68K to an address saved in the floating-point state area (see below). This address should refer to a subprogram in the user's code space. When the halt routine is invoked, the top of the stack is a word containing the number of bytes of parameters (including the return address) on the stack when FP68K was originally called. Beyond that word is the exact stack frame from when FP68K was originally called.

ELEMS68K has no built—in halt mechanism, though a subsidiary FP68K operation may halt.

State area

ELEMS68K maintains no static state. FP68K maintains 3 words of static state across invocations. The first word contains mode and flag bits, much like the CPU Status Register. The next long word is the user trap address. There are two important issues: where is the state area and how is it initialized?

The state area may be a fixed area in memory, as in MAC, or at a fixed offset from a register like A6, as in LISA, or in some user area if FP68K is linked as a subroutine. The state area may even be kept within FP68K itself, though this makes the code self-modifying and thus NON-REENTRANT.

In multi-process environments, care must be taken to see that different state areas are kept for the different processes (again, think of the CPU Status Register). For example, if the state area is kept in a fixed location in memory, it must be swapped each time a new process is swapped in.

The location of the state area must be known at ASSEMBLY TIME. As indicated in the programmer's guide document, the code must be set up for the particular host environment.

When a new process is started up, the state area must be initialized. Fortunately, this is easy. Just clear to 0 the first word of the state area (i.e. the mode and flag word).

CPU Condition Code Register

The Comparison operation leaves the CCR in a well-defined state. After Comparison, the CCR is set for a conditional branch, although the flags are used in a way different from the integer CPU comparisons; see the "User's Guide" for details.

CPU Register DO

The Remainder operation leaves the low-order integer quotient (between -127 and +127) in DO.W. The high half of DO.L is undefined. This intrusion into the register file is extremely valuable in argument reduction -- the principal use of Remainder. The state of DO after an invalid remainder is

undefined.

SANE

There is a SANE (Standard Apple Numeric Environment) library of utility functions based on FP68K, as well as a corresponding Elems library based on ELEMS68K. These libraries are supported on Apple III Pascal systems as well. The library provides access to the package from (Lisa) Pascal. Aside from support of basic arithmetic and elementary functions, the utilities manipulate the modes and flags and provide ASCII <--> floating-point conversions. All applications software should use this package because of its high degree of portability.

Assembly language programmers will invoke FP68K and ELEMS68K directly but will depend on some library for routines to convert between ASCII strings and the canonical decimal format which FP68K recognizes. A set of mnemonic MACROS has been provide to expedite assembly coding.

Compiling Pascal programs

A Pascal program which exploits the SANE and Elems interfaces must include lines such as

uses {\$U <some volume>:SANE.OBJ} SANE; uses {\$U <some volume>:ELEMS.OBJ} Elems; in order to gain access to the types and procedures defined there. Then the program must be linked with SANE.OBJ and ELEMS.OBJ (the Pascal parts of the interface), as well as SANEASM.OBJ and ELEMSASM.OBJ (the assembly language parts of the interface).

Pascal procedures

Programmers should consult the INTERFACE section of the SANE and Elems interfaces (files SANE.TEXT and ELEMS.TEXT) in the following pages. This interface reflects the architecture discussed in the "User's Guide". It is two-address, with the destination operand in the extended format except for format conversions conversions.

Macros

A set of macros provides direct contact with the arithmetic package, using the interface described in the "User's Guide". The macros take care of the opcode and the JSR, but the programmer must explicity push the required argument addresses. The macros do not take effective address arguments and push them itself because of the problems that arise if the destination operand is given as an offset from SP (which changes when the first operand address is pushed). The macros are listed after the Pascal interface.

Sample program

The test programs ITxxx.TEXT, IOxxx.TEXT, and TDFP.TEXT provide a nontrivial view of how to use the Pascal interface to FP68K and ELEMS68K.

```
{ he ''SANE Interface'' }
{^fo '28 December 1982'Page %'Apple Confidential' }
{$C Copyright Apple Computer, 1982 }
{MacIntosh version.}
UNIT Sane;
   INTERFACE
       CONST
           SIGDIGLEN = 20; { Maximum length of SigDig. }
           DECSTRLEN = 80; { Maximum length of DecStr. }
       TYPE
           ** Numeric types.
               Single = array [0..1] of integer;
              Double = array [0..3] of integer;
              Comp = array [0..3] of integer;
              Extended = array [0..4] of integer;
           ** Decimal string type and intermediate decimal type,
           ** representing the value:
           ** (-1)^sgn * 10^exp * dig
               SigDig = string [SIGDIGLEN];
               DecStr = string [DECSTRLEN];
               Decimal = record
                            sgn : 0..1; { Sign (0 for pos, 1 for neg). }
                            exp : integer; { Exponent. }
                            sig : SigDig { String of significant digits. }
{ ne 16 }
           ** Modes, flags, and selections.
           ** NOTE: the values of the style element of the DecForm record
           ** have different names from the PCS version to avoid name
           ** conflicts.
               Environ = integer;
               RoundDir = (TONEAREST, UPWARD, DOWNWARD, TOWARDZERO);
                        = (GT, LT, GL, EQ, GE, LE, GEL, UNORD);
               Re10p
                         { > < <> = >= <= <=>}
               Exception = (INVALID, UNDERFLOW, OVERFLOW, DIVBYZERO,
                                                                INEXACT);
```

```
NumClass = (SNAN, QNAN, INFINITE, ZERO, NORMAL, DENORMAL);
               DecForm = record
                               style : (FloatDecimal, FixedDecimal);
                               digits : integer
                           end;
{^ne 35 }
                 ------
       ** Two address, extended-based arithmetic operations.
       procedure AddS (x : Single; var y : Extended);
       procedure AddD (x : Double; var y : Extended);
       procedure AddC (x : Comp; var y : Extended);
       procedure AddX (x : Extended; var y : Extended);
           {y := y + x}
       procedure SubS (x : Single; var y : Extended);
       procedure SubD (x : Double; var y : Extended);
       procedure SubC (x : Comp; var y : Extended);
       procedure SubX (x : Extended; var y : Extended);
           \{ y := y - x \}
       procedure MulS (x : Single; var y : Extended);
procedure MulD (x : Double; var y : Extended);
       procedure MulC (x : Comp; var y : Extended);
       procedure MulX (x : Extended; var y : Extended);
           {y := y * x}
       procedure DivS (x : Single; var y : Extended);
       procedure DivD (x : Double; var y : Extended);
       procedure DivC (x : Comp; var y : Extended);
       procedure DivX (x : Extended; var y : Extended);
           \{ y := y / x \}
       function CmpX (x : Extended; r : RelOp;
                                     y : Extended) : boolean;
           {xry}
       function RelX (x, y : Extended) : RelOp;
           { x RelX y, where RelX in [GT, LT, EQ, UNORD] }
{^ne 18 }
       {-----
       ** Conversions between Extended and the other numeric types,
       ** including the types integer and Longint.
       procedure I2X (x : integer; var y : Extended);
       procedure S2X (x : Single; var y : Extended);
procedure D2X (x : Double; var y : Extended);
       procedure C2X (x : Comp; var y : Extended);
       procedure X2X (x : Extended; var y : Extended);
           { y := x (arithmetic assignment) }
```

```
procedure X2I (x : Extended; var y : integer);
       procedure X2S (x : Extended; var y : Single);
       procedure X2D (x : Extended; var y : Double);
       procedure X2C (x : Extended; var y : Comp);
          { y := x (arithmetic assignment) }
{^ne 9 }
                         ** These conversions apply to 68K systems only. Longint is
       ** a 32-bit two's complement integer.
       ____}
       procedure L2X (x : Longint; var y : Extended);
       procedure X2L (x : Extended; var y : Longint);
          { y := x (arithmetic assignment) }
{^ne 17 }
       ** Conversions between the numeric types and the intermediate
       ** decimal type.
       procedure S2Dec (f : DecForm; x : Single; var y : Decimal);
       procedure D2Dec (f : DecForm; x : Double; var y : Decimal);
procedure C2Dec (f : DecForm; x : Comp; var y : Decimal);
       procedure X2Dec (f : DecForm; x : Extended; var y : Decimal);
           { y := x (according to the format f) }
       procedure Dec2S (x : Decimal; var y : Single);
       procedure Dec2D (x : Decimal; var y : Double);
       procedure Dec2C (x : Decimal; var y : Comp);
       procedure Dec2X (x : Decimal; var y : Extended);
          \{ y := x \}
{^ne 18 }
       ** Conversions between the numeric types and strings.
       ** (These conversions have a built-in scanner/parser to convert
       ** between the intermediate decimal type and a string.)
       _____
       procedure S2Str (f : DecForm; x : Single; var y : DecStr);
       procedure D2Str (f : DecForm; x : Double; var y : DecStr);
       procedure C2Str (f : DecForm; x : Comp; var y : DecStr);
       procedure X2Str (f : DecForm; x : Extended; var y : DecStr);
           { y := x (according to the format f) }
       procedure Str2S (x : DecStr; var y : Single);
       procedure Str2D (x : DecStr; var y : Double);
       procedure Str2C (x : DecStr; var y : Comp);
       procedure Str2X (x : DecStr; var y : Extended);
           \{ y := x \}
```

```
{^ne 31 }
        ** Numerical 'library' procedures and functions.
        procedure RemX (x : Extended; var y : Extended;
                                                           var quo : integer);
            { new y := remainder of ((old y) / x), such that
                      |\text{new y}| \leq |\mathbf{x}| / 2;
              quo := low order seven bits of integer quotient y / x,
                     so that -127 \le quo \le 127.
        procedure SqrtX (var x : Extended);
            \{x := sqrt(x)\}
        procedure RintX
                          (var x : Extended);
            { x := rounded value of x }
        procedure NegX (var x : Extended);
            \{ x := -x \}
        procedure AbsX
                        (var x : Extended);
            \{ x := |x| \}
        procedure CpySgnX (var x : Extended; y : Extended);
            { x := x with the sign of y }
        procedure NextS (var x : Single; y : Single);
        procedure NextD (var x : Double; y : Double);
        procedure NextX (var x : Extended; y : Extended);
            { x := next representable value from x toward y }
        function ClassS (x : Single; var sgn : integer) : NumClass;
        function ClassD (x : Double; var sgn : integer) : NumClass; function ClassC (x : Comp; var sgn : integer) : NumClass;
        function ClassX (x : Extended; var sgn : integer) : NumClass;
            { sgn := sign of x (0 for pos, 1 for neg) }
        procedure ScalbX (n : integer; var y : Extended);
            {y := y * 2^n}
        procedure LogbX (var x : Extended);
            { returns unbiased exponent of x }
{ ne 16 }
        ** Manipulations of the static numeric state.
        procedure SetRnd (r : RoundDir);
        procedure SetEnv (e : Environ);
        procedure ProcExit(e : Environ);
        function GetRnd : RoundDir;
        procedure GetEnv (var e : Environ);
        procedure ProcEntry (var e : Environ);
        function TestXcp (x : Exception) : boolean;
        procedure SetXcp (x : Exception; OnOff : boolean);
        function TestHlt (x : Exception) : boolean;
        procedure SetHlt (x : Exception; OnOff : boolean);
```

^sp	32767	}			235 CO ME MAN ALC MAT ACC	n and see and and see and an	a mana gilato dunha kulum vime suna amba calab 4000 kilo	o 2000 2000 1000 2000 1000 1000	** *** 6 ℃ 6 ≫ 9 **
]	MPLEME	ENTATION							
	{\$1	SANEIMI	P.TEXT}						
D									

```
{$C Copyright Apple Computer Inc., 1983 }
UNIT Elems;
    { Macintosh version. }
    INTERFACE
        USES
            {$U OBJ:SANE.OBJ }
                SANE { Standard Apple Numeric Environment } ;
        procedure Log2X (var x : Extended);
            \{ x := log2(x) \}
        procedure LnX (var x : Extended);
            \{ x := ln (x) \}
        procedure LnlX (var x : Extended);
            \{ x := ln (l + x) \}
        procedure Exp2X (var x : Extended);
            \{ x := 2^x \}
        procedure ExpX (var x : Extended);
            \{ x := e^x \}
        procedure ExplX (var x : Extended);
            \{ x := e^x - 1 \}
        procedure XpwrI (i : integer; var x : Extended);
            \{ x := x^i \}
        procedure XpwrY (y : Extended; var x : Extended);
            \{ x := x^y \}
        procedure Compound (r, n : Extended; var x : Extended);
            \{ x := (1 + r)^n \}
        procedure Annuity (r, n : Extended; var x : Extended);
            \{ x := (1 - (1 + r)^{-n}) / r \}
        procedure SinX (var x : Extended);
            \{ x := sin(x) \}
        procedure CosX (var x : Extended);
            \{ x := cos(x) \}
        procedure TanX (var x : Extended);
            \{ x := tan(x) \}
```

```
procedure AtanX (var x : Extended);
      \{ x := atan(x) \}
   procedure NextRandom (var x : Extended);
      \{ x := next random (x) \}
IMPLEMENTATION
                                                  EXTERNAL:
   procedure Log2X { (var x : Extended) } ;
   procedure LnX { (var x : Extended) } ;
                                                  EXTERNAL;
   procedure LnlX { (var x : Extended) } ;
                                                  EXTERNAL:
   procedure Exp2X { (var x : Extended) } ;
                                                  EXTERNAL;
   procedure ExpX { (var x : Extended) } ;
                                                  EXTERNAL;
                                                  EXTERNAL;
   procedure ExplX { (var x : Extended) } ;
   {
      Since Elems implementation expects pointer to integer argument,
      use this extra level of interface.
   procedure XpwrIxxx(var i : integer; var x : Extended);
                                                  EXTERNAL;
   procedure XpwrI { (i : integer; var x : Extended) };
   begin
          XpwrIxxx(i, x);
   end;
   procedure XpwrY { (y : Extended; var x : Extended) } ; EXTERNAL;
   procedure Compound { (r, n : Extended; var x : Extended) } ; EXTERNAL;
   procedure Annuity { (r, n : Extended; var x : Extended) } ; EXTERNAL;
   procedure SinX { (var x : Extended) } ;
                                                  EXTERNAL;
   procedure CosX { (var x : Extended) } ;
                                                  EXTERNAL;
   procedure TanX { (var x : Extended) };
                                                  EXTERNAL;
   procedure AtanX { (var x : Extended) } ;
                                                  EXTERNAL;
   procedure NextRandom { (var x : Extended) };
                                                  EXTERNAL;
END
```

These macros give assembly language access to the Mac floating-point arithmetic routines. The arithmetic has just one entry point. It is typically accessed through the tooltrap FP68K, although a custom version of the package may be linked as an object file, in which case the entry point is the label %FP68K.

All calls to the arithmetic take the form: PEA <source address> <destination address> MOVE.W <opcode>,-(SP)

FP68K

All operands are passed by address. The <opcode> word specifies the instruction analogously to a 68000 machine instruction. Depending on the instruction, there may be from one to three operand addresses passed.

This definition file specifies details of the <opcode> ; word and the floating point state word, and defines some handy macros.

Modification history:

29AUG82: WRITTEN BY JEROME COONEN

130CT82: FB CONSTRANTS ADDED (JTC)

28DEC82: LOGB, SCALB ADDED, INF MODES OUT (JTC).

29APR83: ABS, NEG, CPYSGN, CLASS ADDED (JTC).

03MAY83: NEXT, SETXCP ADDED (JTC).

28MAY83: ELEMENTARY FUNCTIONS ADDED (JTC).

04JUL83: SHORT BRANCHES, TRIG AND RAND ADDED (JTC).

01NOV83: PRECISION CONTROL MADE A MODE (JTC).

; This constant determines whether the floating point unit ; is accessed via the system dispatcher after an A-line ; trap, or through a direct subroutine call to a custom ; version of the package linked directly to the application.

• EQU ATRAP 0 ;0 for JSR and 1 for A-line BTRAP ;0 for JSR and 1 for A-line • EQU

> .MACRO JSRFP · IF

ATRAP

FP68K

.ELSE

· REF FP68K FP68K

JSR

. ENDC . ENDM

```
.MACRO JSRELEMS
.IF BTRAP
_ELEMS68K
.ELSE
.REF ELEMS68K
JSR ELEMS68K
.ENDC
.ENDM
```

; OPERATION MASKS: bits \$001F of the operation word ; determine the operation. There are two rough classes of ; operations: even numbered opcodes are the usual ; arithmetic operations and odd numbered opcodes are non-; arithmetic or utility operations. \$0000 . EQU FOADD .EQU \$0002 FOSUB \$0004 . EQU FOMUL .EQU \$0006 FODIV \$0008 . EQU FOCMP \$000A . EQU FOCPX \$000C . EQU FOREM .EQU \$000E FOZ2X . EQU \$0010 FOX2Z .EQU \$0012 FOSQRT \$0014 . EQU FORTI .EQU \$0016 FOTTI .EQU \$0018 FOSCALB .EQU \$001A FOLOGB .EQU \$001C FOCLASS ; UNDEFINED \$001E .EQU . EQU \$0001 FOSETENV .EQU \$0003 FOGETENV \$0005 . EQU FOSETTV . EQU \$0007 FOGETTV \$0009 . EQU FOD2B \$000B . EQU FOB2D \$000D . EQU FONEG .EQU \$000F FOABS \$0011 . EQU FOCPYSGNX .EQU \$0013 FONEXT \$0015 . EQU FOSETXCP .EQU \$0017 FOPROCENTRY \$0019 .EQU FOPROCEXIT \$001B FOTESTXCP .EQU • EQU \$001D ; UNDEFINED .EQU \$001F : UNDEFINED

```
; OPERAND FORMAT MASKS: bits $3800 determine the format of
; any non-extended operand.
            •EQU $0000 ; extended -- 80-bit float
•EQU $0800 ; double -- 64-bit float
•EQU $1000 ; single -- 32-bit float
•EQU $2000 ; integer -- 16-bit integer
FFEXT
FFDBL
FFSGL
FFINT
FFLNG
              •EQU $2800 ; long int -- 32-bit integer
FFCOMP
              •EQU $3000 ; accounting -- 64-bit int
; Bit indexes for error and halt bits and rounding modes in
; the state word. The word is broken down as:
        $8000 -- unused
        $6000 -- rounding modes
                 $0000 -- to nearest
                 $2000 -- toward +infinity
                 $4000 -- toward -infinity
                 $6000 -- toward zero
        $1F00 -- error flags
                 $1000 -- inexact
                 $0800 -- division by zero
                 $0400 -- overflow
                 $0200 -- underflow
                 $0100 -- invalid operation
        $0080 -- result of last rounding
                 $0000 -- rounded down in magnitude
                 $0080 -- rounded up in magnitude
        $0060 -- precision control
                 $0000 -- extended
                 $0020 -- double
                 $0040 -- single
                 $0060 -- ILLEGAL
        $001F -- halt enables, corresponding to error flags
; The bit indexes are based on the byte halves of the state
; word.
```

¹ November 83

FBSGL

.EQU 6 ; single precision control

```
; FLOATING CONDITIONAL BRANCHES: floating point comparisons
; set the CPU condition code register (the CCR) as follows:
       relation X N Z V C
       equal 0 0 1 0 0 less than 1 1 0 0 1
       greater than 0 0 0 0 0 0 unordered 0 0 0 1 0
; The macros below define a set of so-called floating
; branches to spare the programmer repeated refernces to the
; the table above.
        .MACRO FBEQ
        BEO
                %1
        . ENDM
        .MACRO FBLT
        BCS
                %1
        . ENDM
        .MACRO FBLE
        BLS
                %1
        . ENDM
        .MACRO FBGT
        BGT
                %1
        . ENDM
        .MACRO FBGE
        BGE
                %1
        . ENDM
        .MACRO FBULT
        BLT
                %1
        . ENDM
        .MACRO FBULE
        BLE
                 %1
        .ENDM
        .MACRO FBUGT
        BHI
                 %1
        . ENDM
        .MACRO FBUGE
        BCC
                 %1
```

. ENDM

BVS

.MACRO FBU

%1

.ENDM

•MACRO FBO BVC %1

. ENDM

•MACRO FBNE BNE %1

. ENDM

•MACRO FBUE

BEQ %1 BVS %1

. ENDM

.MACRO FBLG

%1

BNE

BVC %1

. ENDM

; Short branch versions.

.MACRO FBEQS

BEQ.S %1

. ENDM

.MACRO FBLTS

BCS.S %1

. ENDM

.MACRO FBLES

BLS.S %1

. ENDM

•MACRO FBGTS

BGT.S %1

. ENDM

.MACRO FBGES

BGE.S %1

. ENDM

.MACRO FBULTS

BLT.S %1

. ENDM

.MACRO FBULES

BLE.S %1

. ENDM

.MACRO FBUGTS

BHI.S %1

. ENDM

.MACRO FBUGES BCC.S %1 . ENDM .MACRO FBUS BVS.S . ENDM .MACRO FBOS BVC.S . ENDM .MACRO FBNES BNE.S %1 . ENDM .MACRO FBUES BEQ.S %1 BVS.S %1 . ENDM .MACRO FBLGS BNE.S %1 BVC.S %1 . ENDM

OPERATION MACROS:

THESE MACROS REQUIRE THAT THE OPERANDS' ADDRESSES FIRST BE PUSHED ON THE STACK. THE MACROS CANNOT THEMSELVES PUSH THE ADDRESSES SINCE THE ADDRESSES MAY BE SP-RELATIVE, IN WHICH CASE THEY REQUIRE PROGRAMMER CARE.

; OPERATION MACROS: operand addresses should already be on ; the stack, with the destination address on top. The ; suffix X, D, S, or C determines the format of the source ; operand -- extended, double, single, or computational ; respectively; the destination operand is always extended.

; Addition.

•MACRO FADDX

MOVE.W #FFEXT+FOADD, -(SP)

JSRFP

. ENDM

.MACRO FADDD

MOVE.W #FFDBL+FOADD, -(SP)

JSRFP

. ENDM

```
.MACRO FADDS
       MOVE.W #FFSGL+FOADD,-(SP)
       JSRFP
        . ENDM
        •MACRO FADDC
       MOVE.W #FFCOMP+FOADD,-(SP)
       JSRFP
        . ENDM
 Subtraction.
        MACRO FSUBX
       MOVE.W #FFEXT+FOSUB, -(SP)
       JSRFP
        . ENDM
        •MACRO FSUBD
       MOVE.W #FFDBL+FOSUB,-(SP)
       JSRFP
        . ENDM

    MACRO FSUBS

       MOVE.W #FFSGL+FOSUB,-(SP)
       JSRFP
        . ENDM
        .MACRO FSUBC
       MOVE.W #FFCOMP+FOSUB,-(SP)
       JSRFP
        . ENDM
; Multiplication.
        MACRO FMULX
       MOVE.W #FFEXT+FOMUL,-(SP)
       JSRFP
        . ENDM
        .MACRO FMULD
       MOVE.W #FFDBL+FOMUL,-(SP)
       JSRFP
        . ENDM
        .MACRO FMULS
       MOVE.W #FFSGL+FOMUL,-(SP)
       JSRFP
       . ENDM
```

.MACRO FMULC

```
MOVE.W #FFCOMP+FOMUL,-(SP)
       JSRFP
        . ENDM
; Division.
        .MACRO FDIVX
       MOVE.W #FFEXT+FODIV,-(SP)
       JSRFP
        . ENDM
        .MACRO FDIVD
       MOVE.W #FFDBL+FODIV,-(SP)
       JSRFP
        . ENDM
        .MACRO FDIVS
       MOVE.W #FFSGL+FODIV,-(SP)
       JSRFP
        . ENDM
        .MACRO FDIVC
       MOVE.W #FFCOMP+FODIV,-(SP)
       JSRFP
        . ENDM
; Compare, signaling no exceptions.
        .MACRO FCMPX
       MOVE.W #FFEXT+FOCMP, -(SP)
       JSRFP
        . ENDM
        .MACRO FCMPD
       MOVE.W #FFDBL+FOCMP,-(SP)
        JSRFP
        . ENDM
        .MACRO FCMPS
       MOVE.W #FFSGL+FOCMP,-(SP)
        JSRFP
        . ENDM
        .MACRO FCMPC
        MOVE.W #FFCOMP+FOCMP,-(SP)
        JSRFP
        . ENDM
```

```
; Compare, signaling invalid operation if the two operands
; are unordered.
        MACRO FCPXX
        MOVE.W #FFEXT+FOCPX,-(SP)
        JSRFP
        . ENDM

    MACRO FCPXD

        MOVE.W #FFDBL+FOCPX,-(SP)
        JSRFP
        . ENDM
        .MACRO FCPXS
        MOVE.W #FFSGL+FOCPX,-(SP)
        JSRFP
        . ENDM
        •MACRO FCPXC
        MOVE.W #FFCOMP+FOCPX,-(SP)
        JSRFP
```

; Remainder. The remainder is placed in the destination, ; and the low bits of the integer quotient are placed in ; the low word of register DO.

```
•MACRO FREMX
MOVE.W #FFEXT+FOREM,-(SP)
JSRFP
. ENDM
•MACRO FREMD
MOVE.W #FFDBL+FOREM,-(SP)
JSRFP
ENDM
•MACRO FREMS
MOVE.W #FFSGL+FOREM, -(SP)
JSRFP
ENDM
.MACRO FREMC
MOVE.W #FFCOMP+FOREM, -(SP)
JSRFP
. ENDM
```

. ENDM

; Compare the source operand to the extended format and ; place in the destination.

```
MACRO FX2X
       MOVE.W #FFEXT+FOZ2X,-(SP)
       JSRFP
       . ENDM
       •MACRO FD2X
       MOVE.W #FFDBL+FOZ2X,-(SP)
       JSRFP
       .ENDM
       .MACRO FS2X
       MOVE.W #FFSGL+FOZ2X,-(SP)
       JSRFP
       . ENDM
       .MACRO FI2X
                                       ; 16-bit integer
       MOVE.W #FFINT+FOZ2X,-(SP)
       JSRFP
       . ENDM
       .MACRO FL2X
                                       ; 32-bit integer
       MOVE.W #FFLNG+FOZ2X,-(SP)
       JSRFP
       . ENDM
       .MACRO FC2X
       MOVE.W #FFCOMP+FOZ2X,-(SP)
       JSRFP
       . ENDM
; Convert the extended source operand to the specified
; format and place in the destination.
       MACRO FX2D
       MOVE.W #FFDBL+FOX2Z,-(SP)
       JSRFP
       . ENDM
       MACRO FX2S
       MOVE.W #FFSGL+FOX2Z,-(SP)
       JSRFP
       . ENDM
       .MACRO FX2I
                                       ; 16-bit integer
       MOVE.W #FFINT+FOX2Z,-(SP)
       JSRFP
       . ENDM
       .MACRO FX2L
                                       ; 32-bit integer
       MOVE.W #FFLNG+FOX2Z,-(SP)
       JSRFP
```

```
    MACRO FX2C

        MOVE.W #FFCOMP+FOX2Z,-(SP)
        JSRFP
        . ENDM
; Miscellaneous operations applying only to extended
; operands. The input operand is overwritten with the
; computed result.
; Square root.
        MACRO FSORTX
        MOVE.W #FOSQRT,-(SP)
        JSRFP
        . ENDM
; Round to integer, according to the current rounding mode.
        MACRO FRINTX
        MOVE.W #FORTI,-(SP)
        JSRFP
        . ENDM
; Round to integer, forcing rounding toward zero.
        .MACRO FTINTX
        MOVE.W #FOTTI, -(SP)
        JSRFP
        . ENDM
; Set the destination to the product:
; (destination) * 2^(source)
; where the source operand is a 16-bit integer.

    MACRO FSCALBX

        MOVE.W #FFINT+FOSCALB,-(SP)
        JSRFP
        . ENDM
; Replace the destination with its exponent, converted to
; the extended format.
        .MACRO FLOGBX
       MOVE.W #FOLOGB, -(SP)
        JSRFP
        . ENDM
; Non-arithmetic sign operations on extended operands.
; Negate.

    MACRO FNEGX
```

1 November 83

. ENDM

```
MOVE.W #FONEG, -(SP)
        JSRFP
        . ENDM
; Absolute value.
        .MACRO FABSX
       MOVE.W #FOABS, -(SP)
       JSRFP
        . ENDM
; Copy the sign of the destination operand onto the sign of
; the source operand. Note that the source operand is
; modified.
        .MACRO FCPYSGNX
       MOVE.W #FOCPYSGN, -(SP)
        JSRFP
        . ENDM
; The nextafter operation replaces the source operand with
; its nearest representable neighbor in the direction of the
; destination operand. Note that both operands are of the
; the same format, as specified by the usual suffix.
        .MACRO FNEXTS
        MOVE.W #FFSGL+FONEXT,-(SP)
        JSRFP
        . ENDM
        MACRO FNEXTD
        MOVE.W #FFDBL+FONEXT,-(SP)
        JSRFP
        . ENDM
        .MACRO FNEXTX
        MOVE.W #FFEXT+FONEXT,-(SP)
        JSRFP
        . ENDM
; The classify operation places an integer in the
; destination. The sign of the integer is the sign of the
; source. The magnitude is determined by the value of the
; source, as indicated by the equates.
```

```
.MACRO FCLASSS
MOVE.W #FFSGL+FOCLASS,-(SP)
JSRFP
. ENDM
.MACRO FCLASSD
MOVE.W #FFDBL+FOCLASS,-(SP)
JSRFP
. ENDM
MACRO FCLASSX
MOVE.W #FFEXT+FOCLASS,-(SP)
JSRFP
. ENDM
•MACRO FCLASSC
MOVE.W #FFCOMP+FOCLASS,-(SP)
JSRFP
. ENDM
```

; These four operations give access to the floating point; state (or environment) word and the halt vector address; The sole input operand is a pointer to the word or address; to be placed into the arithmetic state area or read from; it.

```
MOVE.W #FOGETENV,-(SP)
JSRFP
.ENDM

.MACRO FSETENV
MOVE.W #FOSETENV,-(SP)
JSRFP
.ENDM

.MACRO FGETTV
MOVE.W #FOGETTV,-(SP)
JSRFP
.ENDM

.MACRO FSETTV
MOVE.W #FOSETTV,-(SP)
JSRFP
```

MACRO FGETENV

; Both FPROCENTRY and FPROCEXIT have one operand — a ; pointer to a word. The entry procedure saves the current ; floating point state in that word and resets the state ; to 0, that is all modes to default, flags and halts to ; OFF. The exit procedure performs the sequence:

. ENDM

```
1. Save current error flags in a temporary.
       2. Restore the state saved at the address given by
               the parameter.
       3. Signal the exceptions flagged in the temporary,
               halting if so specified by the newly
               restored state word.
; These routines serve to handle the state word dynamically
; across subroutine calls.
       .MACRO FPROCENTRY
       MOVE.W #FOPROCENTRY, -(SP)
       JSRFP
       . ENDM
       .MACRO FPROCEXIT
       MOVE.W #FOPROCEXIT, -(SP)
       JSRFP
       . ENDM
; FSETXCP is a null arithmetic operation which stimulates
; the indicated exception. It may be used by library
; routines intended to behave like elementary operations.
; The operand is a pointer to an integer taking any value
; between FBINVALID and FBINEXACT.
; FTESTXCP tests the flag indicated by the integer pointed
; to by the input address. The integer is replaced by a
; Pascal boolean (word $0000=false, $0100=true)
MACRO FSETXCP
       MOVE.W #FOSETXCP,-(SP)
       JSRFP
       ENDM
       .MACRO FTESTXCP
       MOVE.W #FOTESTXCP,-(SP)
       JSRFP
       . ENDM
; WARNING: PASCAL ENUMERATED TYPES, LIKE THOSE OF THE
; DECIMAL RECORD, ARE STORED IN THE HIGH-ORDER BYTE OF THE
; ALLOCATED WORD, IF POSSIBLE. THUS THE SIGN HAS THE
; INTEGER VALUE O FOR PLUS AND 256 (RATHER THAN 1)
; FOR MINUS.
; BINARY-DECIMAL CONVERSION: The next routines convert
; between a canonical decimal format and the binary format
; specified. The decimal format is defined in Pascal as
   CONST
       SIGDIGLEN = 20;
```

```
TYPE
       SigDig = string [SIGDIGLEN];
       Decimal = record
                      sgn : 0..1;
                      exp : integer;
                      sig : SigDig
                  end;
; Note that Lisa Pascal stores the sgn in the high-order
; byte of the allotted word, so the two legal word values
; of sgn are 0 and 256.
; Decimal to binary conversion is governed by a format
; record defined in Pascal as:
   TYPE
       DecForm = record
                      style : (FloatDecimal, FixedDecimal);
                      digits : integer
                  end;
; Note again that the style field is stored in the high-
; order byte of the allotted word.
; These are the only operations with three operands. The
; pointer to the format record is deepest in the stack,
; then the source pointer, and finally the destination
; pointer.
        .MACRO FDEC2X
        MOVE.W #FFEXT+FOD2B,-(SP)
        JSRFP
        . ENDM
        .MACRO FDEC2D
        MOVE.W #FFDBL+FOD2B, -(SP)
        JSRFP
        . ENDM
        MACRO FDEC2S
        MOVE.W #FFSGL+FOD2B,-(SP)
        JSRFP
        . ENDM
        MACRO FDEC2C
        MOVE.W #FFCOMP+FOD2B,-(SP)
        JSRFP
        . ENDM
```

```
; Binary to decimal conversion.
        .MACRO FX2DEC
        MOVE.W #FFEXT+FOB2D,-(SP)
        JSRFP
        . ENDM
        .MACRO FD2DEC
        MOVE.W #FFDBL+FOB2D,-(SP)
        JSRFP
        . ENDM
        .MACRO FS2DEC
        MOVE.W #FFSGL+FOB2D,-(SP)
        JSRFP
        . ENDM

    MACRO FC2DEC

        MOVE.W #FFCOMP+FOB2D,-(SP)
        JSRFP
        . ENDM
; Equates and macros for elementary functions.
FOLNX
              • EQU
                        $0000
               .EQU
FOLOG2X
                        $0002
               .EQU
FOLN1X
                        $0004
FOLOG21X
               • EQU
                        $0006
FOEXPX
                • EQU
                        $0008
FOEXP2X
               • EQU
                        $000A
FOEXP1X
               • EQU
                        $000C
FOEXP21X
               • EQU
                        $000E
FOXPWRI
                • EQU
                        $8010
FOXPWRY
               •EQU
                        $8012
```

FOATANX .EQU \$001E
FORANDOMX .EQU \$0020

.MACRO FLNX
MOVE.W #FOLNX,-(SP)
JSRELEMS
.ENDM

.MACRO FLOG2X

• EQU

EQU

. EQU

• EQU

• EQU

\$C014

\$C016

\$0018

\$001A \$001C

MOVE.W #FOLOG2X,-(SP)

FOCOMPOUNDX

FOANNUITYX

FOSINX

FOCOSX

FOTANX

JSRELEMS

. ENDM

.MACRO FLN1X

MOVE.W #FOLN1X,-(SP)

JSRELEMS

. ENDM

•MACRO FLOG21X

MOVE.W #FOLOG21X,-(SP)

JSRELEMS

. ENDM

MACRO FEXPX

MOVE.W #FOEXPX,-(SP)

JSRELEMS

. ENDM

•MACRO FEXP2X

MOVE.W #FOEXP2X,-(SP)

JSRELEMS

. ENDM

.MACRO FEXPlX

MOVE.W #FOEXP1X,-(SP)

JSRELEMS

. ENDM

•MACRO FEXP21X

MOVE.W #FOEXP21X,-(SP)

JSRELEMS

. ENDM

.MACRO FXPWRI

MOVE.W #FOXPWRI,-(SP)

JSRELEMS

. ENDM

.MACRO FXPWRY

MOVE.W #FOXPWRY,-(SP)

JSRELEMS

. ENDM

.MACRO FCOMPOUNDX

MOVE.W #FOCOMPOUNDX,-(SP)

JSRELEMS

. ENDM

.MACRO FANNUITYX

MOVE.W #FOANNUITYX, -(SP)

JSRELEMS

. ENDM

.MACRO FSINX

```
MOVE.W #FOSINX, -(SP)
JSRELEMS
. ENDM
.MACRO FCOSX
MOVE.W #FOCOSX,-(SP)
JSRELEMS
. ENDM
.MACRO FTANX
MOVE.W #FOTANX, -(SP)
JSRELEMS
. ENDM
.MACRO FATANX
MOVE.W #FOATANX, -(SP)
JSRELEMS
.ENDM
.MACRO FRANDOMX
MOVE.W #FORANDOMX, -(SP)
JSRELEMS
. ENDM
```

 ${\tt FP68K}$ provides conversions between the extended floating-point format and three integer formats:

int16 -- 16-bit two's complement
int32 -- 32-bit two's complement
comp64 -- 64-bit two's complement with the reserved value
 hexadecimal 80000000000000.

One Pascal program, ITBATTERY.TEXT, tests all three conversions. This document describes how to use and, if necessary, modify the tests.

Compiling and running

ITBATTERY.TEXT uses the SANE interface (see the "High Level Interface" document, so it must be linked with the SANE object files, as well as with the usual nonarithmetic Pascal run-time libraries (e.g. *MPASLIB on Lisa). The program will simply run to completion, with a Pascal HALT if an error is found; execution time may run to 15 minutes on a Lisa system.

What is tested

Each of the integer formats is tested in two phases. First, a collection of specific extended numbers is converted to the integer format, with tests for correct rounding and signaling of the invalid exception when appropriate. Then a set of

integer --> extended --> integer

conversions is run, with the input and output integers compared for equality. In the case of intl6, all 2^16 cases are run. However exhaustive testing of int32 and comp64 is infeasible so a loop is set up to do 2^16 tests from several starting points.

The most important and rigorous set of tests of FP68K is the set of so-called IEEE test vectors. These tests, developed by the author while at Zilog, are used to test implementations of proposed standard P754. They were donated to the IEEE subcommittee 754 by Zilog Inc., and are now distributed by that subcommittee. The tests have undergone major revision within Apple, thanks especially to Jim Thomas of PCS.

Form of the tests

Each vector is an ascii string describing an operation, operands, and the result. For example, "lincl" is the floating-point number (of the format under consideration) next larger than 1. When "l" is subtracted from "lincl", the result is "lulpl", just one unit in the last place of 1. Written this way, the vectors may be applied to any floating-point format. The tests carefully inspect the nuances of rounding and exception handling. A document is under development to explain in detail the next release of the test vectors, scheduled for early 1983, after some last details of the standard are cleared up.

Files

The test vectors are contained in a family of files by the name of TVxxxx.2.TEXT and TWxxx.2.TEXT. The "2" refers to version 2 of the tests. (Version 1 was based on Draft 8.0 of the standard.) The file TLIST.TEXT is a list of the test file names to be used in any given run of the test. Pascal file TD68.TEXT with unit TD68FP.TEXT actually run the tests. These interface with FP68K exclusively through the SANE interface.

Since the binary <--> conversions within FP68K approximate the mathematical identity operation, they lend themselves to certain types of self-testing. For example, if enough decimal digits are kept, then the conversion

binary --> decimal --> binary

is the identity mapping when results are rounded to nearest. The number of digits required turns out to be 9 for single and 17 for double. A similar test performs the first conversion rounding toward plus infinity and the second rounding toward minus infinity. In this case the final result may differ from the starting value by one unit in the direction of the latter rounding, so the program allows this discrepancy.

This document describes the test files and how they can be run. For details of the underlying error analysis (which is quite subtle) see the paper "Accurate Yet Economical Binary-Decimal Conversions" by J. Coonen.

Test programs

The test programs are:

IOS.TEXT
IOSF.TEXT
IOD.TEXT
IODF.TEXT
IONAN.TEXT
IOPSCAN.TEXT

The letter "S" and "D" distinguishes single and double tests. The IOS.TEXT and IOD.TEXT tests run with both rounding to nearest and the directed roundings. The "F" tests use fixed-format output rather than floating-format output for the intermediate decimal string. The IOPSCAN test is used to check the preformance of the printer and scanner used by SANE68, and included from file SAPSCAN.TEXT. The IONAN test checks the input and output conversion of some 20 stock NANs, and then allows the user to enter any decimal string to be converted to the three formats in three rounding modes. Neither IOPSCAN nor IONAN are self-checking; rather, the user must monitor their output.

The tests cover extreme intervals where the decimal numbers are sparsest and densest with respect to the binary numbers. Sparse intervals have the form $[10^N, 2^n]$ where the endpoints are nearly equal. Dense intervals have the corresponding form $[2^m, 10^M]$.

Running the tests

Each of the programs is compiled and run separately. The programs use the SANE interface. A test will HALT with a suitable diagnostic if the test

fails.

The single format cases are few enough that their tests can be run overnight. However, the double format cases will run essentially forever since the number of interesting cases is so great. A few overnight tests should be sufficient.

Background

The so-called I/O routines for scanning and printing floating-point numbers in decimal form are complicated by subtle numerical issues and nettlesome design decisions. For example, even the simplest, stripped-down conversion routines require over one-third the code space (about 1.3K) of the rest of the FP68K binary floating-point package. With a full parser and formatter, the conversion routines are much larger. And it is unclear whether full routines would be flexible enough for use in different language systems and I/O-intensive applications like Visi-Calc.

Where does the responsibility lie? This note argues that the core conversion routines, which are part of the arithmetic package, should be kept very simple. Above them — somewhere in the system — should be a full scanner and formatter available to languages and applications, but not forced upon them. This would lead to the most efficient use of code space and execution time.

The Sad Truth

Numerical I/O can be monstrous. Since each computer language has its own grammar for floating-point numbers and its own conventions for output format, it almost necessary for each language system on a computer to provide significant I/O support. Unfortunately, this may be layered upon the host system's I/O system. And it is not unusual (Apple III, for example) for a language compiler to use different conversion routines than the I/O system the compiled code utilizes.

In another case, designers of the UNIX operating system attempted to route all conversion through the routines atof(), ascii to floating, and the pair ecvt(), fcvt() for floating and fixed conversion to ascii. But even this fairly clean design has led to VERY complicated software shells around atof, ecvt, and fcvt. Numerical accuracy aside, the complexity of just the character hacking is forbidding.

One problem with the UNIX design lies in its failure to properly divide responsibility for the distinct processes involved in conversion, namely:

- 1. Recognize floating-point strings (in compilers, ...)
- 2. Translate strings to numerical values.
- Determine which output format (fixed or floating) is appropriate for a given value.
- 4. Translate a numerical value to a string.

The utilities atof() and ecvt() provide items 2 and 4. Item 3, printing a number in its "nicest" form is provided in rough form through ecvt(). But recognizing strings is left to each language compiler's lexical scanner. Unfortunately, after a scanner has parsed a floating decimal string, it passes it along to atof() where it is parsed once more.

A Proposal for Change

(1) Support, at the arithmetic level, conversions between each of the available binary floating-point types and one decimal structure describable in Pascal as:

(2) Rigidly specify the format of Decimal·sig for decimal to binary conversions, relying upon a lexical scanner to perform the first parse. The decimal value would depend upon the first character of decrec.dig:

```
'I' --> infinity
'Nxxx...x' --> NAN, with optional ascii hex digits 0-9, A-F, a-f
'O' --> zero
'ddd...d' --> string of digits stripped of leading and trailing zeros
```

The digit string would never be more than 20 digits long. If present, the 20-th digit would indicate the absence of nonzero trailing digits beyond the 20-th (to aid in correct rounding).

(3) Specify decimal output format through a structure like the Pascal:

```
{*
    ** Output format specifier.
    *}
type
    DecForm = record
        style: (float, fixed);
        digits: integer
    end;
```

For "float" conversions, digits is the number of significant digits to be delivered in Decimal.sig. For "fixed" conversions, count is the number of fraction digits to be converted (a negative count suppresses conversion of low-order integer digits).

Sometimes it is desired to print a number in the nicest form possible for a given field width. For example, the string "1.23456789" conveys much more information in 10 characters than does "1.2345e+04". Such conversions are

discussed in the next section.

(4) Provide a scanner and formatter which, if not of most general use, provide models that can be tailored to a particular application. Samples are built into the implementation section of the SANE Pascal interface; they are contained in the file SAPSCAN.TEXT.

Binary --> Decimal

The family of routines:

S2Dec

D2 Dec

X2Dec

C2Dec

provide conversions to the Decimal record format described above. Special cases are keyed by the first character of Decimal.sig:

'0' : zero

'I' : infinity

'N' : not-a-number, followed by optional ascii hex digits; if there are

fewer than four, they are padded on the left with 0's.

'?': overflow of fixed-style format

These must be used with a formatter to produce output strings.

The family:

S2Str

D2Str

X2Str

C2Str

uses the built-in formatter, Dec2Str, to generate ascii string output.

Decimal --> Binary

These conversions are povided by the complementary set of procedures: Dec2S, Dec2D, Dec2X, Dec2C, and Str2S, Str2D, Str2X, Str2C. In the case of the Dec2* conversions, the first character of Decimal.sig indicates special cases as noted above for *2Dec conversions.

Infinity and NAN conversions

Infinity is printed and read as a string of sign characters, "+++++" or "----".

On input, NANs have the general form NAN'xxxx:yyy \dots y'. The x's and y's should be ascii hex digits: 0-9, A-F, a-f. The string portion following NAN may be omitted. The x's are padded on the LEFT with 0's to width 4. The y's are padded on the RIGHT with 0's to the width of the NAN's significant bit

field.

On output, NANs will be printed in the same format. Leading x=0 and trailing y=0 are omitted, but at least one x is printed. If all y=0, then the colon and the y field is dropped.

Any unrecognizable string is converted to a NAN.

Background

Applications like accounting spreadsheets typically need to display floating-point values in decimal form within a field of fixed width. For maximum readability, the output should be in integer or fixed-point format if possible, with floating-point format as a last resort. The idea is to avoid listing small integers in the abominable form 0.1000000000000El reminiscent of computing in the McCarthy era.

The problem

Given a binary floating-point number X and an ascii field F, display X in the "nicest", most informative way within F.

A proposal

- 1. If X may be displayed in a subfield of F, pad X on the left with blanks.
- 2. Display the sign of X only if it is '-'.
- 3. If X is an integer and F is wide enough to accommodate X, then display X as an integer, without a trailing '.'.
- 4. Else if X has nonzero integer and fraction parts and F is wide enough to accommodate at least the integer part of F and its trailing '.', then display X in the fixed-point form ZZZZ.YYYY with as many fraction digits as F will accommodate, up to a maximum of 17 significant digits.
- 5. Else if |X| < 1 and F is wide enough that X may be displayed in the form 0.000002ZZZZ with no more 0s just to the right of the decimal point than digits following those 0s, then display X in that fixed-point form with up to 17 significant digits.
- 6. Finally, if all the above fail, then display X in the floating-point form Z.ZZZZZEYYY with as many significant digits up to 17 as F will accommodate, taking into account the width of the exponent field, including its possible sign. Display the sign of the exponent field only if it is '-'.

An implementation

The above choices depend on detailed knowledge of the magnitude of X. For example, in producing floating-point output, it is necessary to know the number of spaces that will be occupied by the decimal exponent (with sign, it could be 1 to 5) in order to know how many significant digits to which to round X. In the worst case, this could mean several calls upon the low-level conversion routine until the proper output is finally obtained.

One easy way to bypass these problems, and keep the fundamental conversion routine simple, is perform the binary -> decimal conversion in two stages. First convert the binary value X to the SANE decimal form:

```
type
  DecStr = string[DECSTRLEN]; { length is 20 for MAC }
  Decimal = record
    sgn: integer; {0 for +, nonzero for -}
    exp: integer; {as though decimal is at the right of...}
    sig: DecStr
end;
```

If the conversion is performed with rounding toward 0, conversion style = float, and digit count = 19, and if the inexact exception flag is cleared before the conversion, then the 19-digit result may be correctly rounded to the desired width after the ultimate output format is determined. Since no more than 17 digits will ever be displayed (recall that 17 digits suffice to distinguish double format binary numbers), the 19 digits together with the inexact exception flag permit correct rounding.

The second step of the conversion decides, on the basis of the intermediate decimal form, which format is appropriate. Then the decimal value is rounded (in decimal!) and displayed as desired. Note that this scheme has as a happy byproduct the ability to round in the (time-honored?) "add half and chop" manner that is unavailable within Apple arithmetic itself.

In the interest of compatibility of the floating-point arithmetic on Apples II/III and Mac/(Lisa?), the following GRITTY DETAILS were discussed on June 29. This is an update on the decisions made then.

- 1. Distinguishing signaling and quiet NANs: use the leading fraction bit, 0-quiet and 1-signaling.
- 2. Explicit leading bit of extended NANs and INFs: ignore it, that is decide whether NAN or INF on the basis of the fraction bits only.
- 3. Quiet NANs have an 8-bit "indicator field" marked by stars in the following extended format hex mask: XXXX XX** XXXX XXXX. This byte is the low half of the leading word of significant bits. The interpretation of the field is as given page 70 of Apple III Pascal, volume 2, subject to enhancements.
- 4. When two quiet NANs are operands to the operations +, -, *, /, and REM, one or the other of the NANs is output. When the indicator fields differ, the NAN with the larger indicator field prevails; ties are broken arbitrarily.
- 5. True to the standard, the sign of an output NAN is unspecified.
- 6. Signaling NANs precipitate the invalid operation exception when they appear as operands.
- 7. Underflow is tested before rounding. CHANGE: this may change depending on P754 deliberations in the late summer of '82
- 8. Projective INF follows the same rules of signs as affine INF. The ABSOLUTELY ONLY differences between affine and projective modes are: the UNORDERED-ness of projective INF in comparisons with finite numbers, and the invalid operation exception that arises from the sum of two projective INFs with the same sign. CHANGE: projective mode may be removed from P754 in late summer '82.
- 9. Treatment of unnormalized extended numbers may differ between systems. 68K implementations will normalize all such, as is expected of the Motorola and Zilog chips. 6502 implementations may support the ANTIQUE warning mode in preliminary releases, though it may never be documented for general consumption.
- $10 \cdot$ The bottom of the extended exponent range is as in the Motorola and Zilog implementations (as opposed to Intel). That is, there is no redundancy between the bottom two exponent values.
- 11. The exponent bias in extended is hex 3FFF, which is used by Intel, Zilog, and Motorola. Motorola may insert a word of garbage between the sign/exp fields and the significant bits in order to have a 96-bit data type.
- 12. Comparisons return results according to local system convenience. 68K: return from the floating-point software with the CPU condition codes set appropriately for a conditional branch. 6502: for lack of a rich set of conditional branches, let the comparison operation be a family of boolean tests like "Is X \leq Y?" The difference between the two systems should be

hidden well below the high-level language interfaces.

- 13. Auxiliary functions: relegate functions like nextafter() to the system numerical library rather than putting them in the arithmetic engine.
- 14. The data types specified by SANE are intl6, comp32, comp64, f32, f64, x80. 68K systems will require int32 as well.
- 15. Is the Pascal assignment: X := Y; an arithmetic operation when both X and Y are variables of the same floating-point format? Or is a straight byte copy sufficient? This is really a language issue one left dangling by the standard. The arithmetic units, if asked to perform a floating move between two floating entities of the same format, will perform a full-blown arithmetic operation. This will cause side effects if the floating value is a signaling NAN (invalid operation) or a denormalized number (underflow).
- 16. Precision control is supported by 6502 and 68K packages, but it is available only through assembly language it is intended only for SPECIAL applications anyway. Precision control implies range control, too.
- 17. There is no "integer overflow" exception.
- 18. Traps? These are so system-dependent there is no hope for perfect consistency. So the issue is left as a local matter for each system. The question relevant to each floating-point engine is: "What information will I be required to spew out in case of a trap?"

TABLE OF CONTENTS

- 1. Design Philosophy
- 2. Data Types
- 3. Arithmetic Operations
- 4. Format Conversions
- 5. Internal Architecture
- 6. External Access
- 7. Calling Sequence
- 8. Comparisons
- 9. Binary-Decimal Conversions
- 10. The State Area
- 11. Traps
- 12. Other Pseudo-Machines
- 13. Arithmetic Abuse
- 14. Size and Performance
- 15. Floating-Point at a Glance ...a graphical view

1. Design Philosophy

The software package FP68K provides binary floating-point arithmetic according to the proposed IEEE standard P754. This arithmetic is in turn the basis for SANE, standard Apple numeric environment. The goal is software compatibility between the various Apple products supporting SANE.

The arithmetic package is reasonably small and fast. Its interface is very simple. And it provides just those operations needed for applications software. Although developed specifically for Mac, the package is designed for use in Lisa, if desired.

The following sequence of examples illustrates the SANE philosophy:

Single operation: X + Y

> P754: Compute as if with unbounded range and precision, then coerce to destination format.

Expression evaluation:
$$Z := (X + Y)/(U + V);$$

SANE: Compute all anonymous intermediate subexpressions to extended, then coerce to destination format.

Loop:

VERY SANE: Wise programmer uses extended variable S to eliminate spurious over/underflows in the inner loop, and to reduce the final rounding error.

Data Types

The arithmetic supports the following data types. All are specified in SANE except for int32 and decimal. Int32 is included for convenience in 68K environments, where 32-bit integers are common. Through the decimal type the package provides the basis for the binary<->decimal conversions required by languages and the I/O system.

int16 -- 16-bit two's-complement integer
int32 -- 32-bit two's-complement integer
comp64 -- 64-bit integer, with one reserved operand value
f32 -- 32-bit single floating-point
f64 -- 64-bit double floating-point
x80 -- 80-bit extended floating-point
decimal -- ascii digit string with integer sign and exponent

3. Arithmetic Operations

These operations apply to floating-point operands:

+, -, *, /, SQRT, REMAINDER, COMPARE,
ROUND TO INTEGER, TRUNCATE TO INTEGER, LOGB, SCALB,
ABSOLUTE VALUE, NEGATE, COPYSIGN, NEXAFTER, CLASS

Except for COMPARE, each produces a floating-point result. COMPARE sets the CPU flag bits according to the two operands. Besides its floating-point result, REMAINDER returns the sign and four least significant bits of its integer quotient in the CPU flags (a very useful trick for argument reduction in the transcendental functions). LOGB replaces a number by is unbiased exponent, in floating form; SCALB scales a number by an integer power of 2.

4. Format Conversions

5. Internal Architecture

The package provides 2-address memory to memory arithmetic operations of the form

 $\langle op \rangle$ DST \longrightarrow DST and SRC $\langle op \rangle$ DST \longrightarrow DST

where DST and SRC are the destination and source operands, respectively. The DST operand is always in the extended format. The conversions have the form:

SRC --> DST

where at least one of SRC and DST is a floating-point format. The package also provides a few support functions in connection with the floating-point error flags and modes.

Extended format results may be coerced to the PRECISION and RANGE of the single or double formats, on an instruction by instruction basis. Then subsequent operations are able to take advantage of the trailing zeros to improve performance. This feature is provided to expedite special-purpose applications such as graphics and is not intended for general use. Only under certain circumstances will it actually obtain a speed advantage, rather than a DISADVANTAGE, since the package is built to do extended arithmetic.

6. External Access

The package is re-entrant, position-independent code, which may be shared in multi-process environments. It is accessed through one entry point, labeled FP68K. Each user process has a static state area consisting of one word of mode bits and error flags, and a two-word halt vector. The package allows for different access to the state word in one-process (Mac) and multi-process (Lisa) environments.

The package preserves all CPU registers across invokations, except that REMAINDER modifies DO. It modifies the CPU condition flags. Except for binary-decimal conversions, it uses little more stack area than is required to save the sixteen 32-bit CPU registers. Since the binary-decimal conversions themselves call the package (to perform multiplies and divides), they use about twice the space of the regular operations.

7. Calling Sequence

A typical invokation of the package will consist of a sequence of four 68K assembly instructions:

PEA	<pre><source address=""/></pre>	"Push Effective Address"
PEA	<pre><destination address=""></destination></pre>	"Push Effective Address"
MOVE.W	<pre><opword>, -(SP)</opword></pre>	;"Push" operation word
JSR	FP68K	;"Call" the package

(If FP68K resides in system memory, the JSR may be replaced by an A-line trap opcode.) Other calls will have more or fewer operand addresses to push onto the stack. The opword is the logical OR of two fields, given here in hexadecimal:

"non-extended" operand format, bits 3800:

0000 -- x80 0800 -- f64 1000 -- f32 1800 -- ILLEGAL 2000 -- int16

```
2800 -- int32
    3000 -- comp64
    3800 -- ILLEGAL
arithmetic operation code, bits 001F:
    0000 -- add
    0002 -- subtract
    0004 -- multiply
    0006 -- divide
    0008 -- compare
    000A -- compare and signal invalid if UNORDERED
    000C -- remainder
    000E -- floating, intxx, comp64 --> extended convert
    0010 -- extended --> intXX, comp64, floating convert
    0012 -- square root
    0014 -- round to integer in floating format
    0016 -- truncate to integer in floating format
    0018 -- scale by integer power of 2
    001A -- replace by unbiased exponent
    001C -- classify the floating input
    001E -- ILLEGAL
    0001 -- put state word
    0003 -- get state word
    0005 -- put halt vector
    0007 -- get halt vector
    0009 -- decimal --> floating convert
    000B -- floating --> decimal convert
    000D -- negate
    000F -- absolute value
    0011 -- copy sign
    0013 -- nextafter
    0015 -- set exception
    0017 -- procedure entry protocol
    0019 -- procedure exit protocol
    001B -- test exception
    001D and 001F are ILLEGAL
```

8. Comparisons

In this arithmetic, comparisons require some extra thought. The trichotomy rule of the real number system — that two numbers are related as LESS, EQUAL, or GREATER — is violated by the NANs, which compare UNORDERED with everything, even themselves. So it is necessary for floating-point comparisons to use the CPU condition codes in a way that seems surprising at first blush:

RELATION	FLAGS:	X	N	Z	V	С
LESS		1	1	0	0	1
EQUAL		0	0	1	0	0
GREATER		0	0	0	0	0
UNORDERED		0	0	0	1	0

This encoding leads to a very convenient mapping between the "floating-point conditional branches" and the CPU conditional branches. In the following table, the '?' refers to UNORDERED. The second column gives the name of the branch macro that provides the "floating branch" (see the "Assembler Support" document).

BRANCH CONDITION	MACRO NOTATION	CPU BRANCH
	FBEQ	BEQ
<	FBLT	BCS
<, =	FBLE	BLS
>	FBGT	BGT
>, =	FBGE	BGE
?, <	FBULT	BLT
?, <, =	FBULE	BLE
?, >	FBUGT	BHI
?, >, =	FBUGE	BCC
? (unordered)	FBU	BVS
<, =, > (ordered)	FBO	BVC
?, <, > (not equal)	FBNE	BNE
?, =	FBUE	BEQ / BVS
<, >	FBLG	BNE / BVC

Only in the last two instances, are two branches required.

The variant comparison instruction, that signals the invalid operation exception if its operands are UNORDERED, is useful in high-level languages since P754 (and SANE) require that certain UNORDERED comparisons be marked invalid.

Further discussion of the language issues of comparisons may be found in "Comparisons and Branching" by Jerome Coonen.

9. Binary-Decimal Conversions

The package provides conversion functions intended to be used in conjunction with scanners and formatters peculiar to the user environment. For decimal to binary conversions, the input parameters are:

```
address of Pascal decimal structure:
    record
        sgn : 0..1;
        exp : integer;
        sig : string[20]
    end;
```

address of target floating variable

The format (f32, f64, x80) of the target is given in the opword. For binary to decimal conversions, the input paramaters are:

```
address of format structure:
    record
    style : (FloatDecimal, FixedDecimal);
    digits: integer
    end;

address of source floating variable

address of decimal structure:
    sign
    exponent
    ascii string of significant digits
```

The interpretation of the latter format element depends on the style of the conversion. For fixed conversions, the digit count gives the number of fraction digits desired (which may be negative). For float conversions, the digit count gives the number of significant digits desired.

Free format binary --> decimal conversions, which display numbers in the "nicest" format possible within given field width constraints, are supported in software, using the float style of conversion. Nice conversions are handy in applications like accounting spreadsheets where tables of numbers are displayed. See the "Binary-Decimal Conversion" document for details. The SANE interface gives details about the decimal format.

10. The State Area

Each user of the package has three words of static floating-point state information. All accesses to the state should be made through the four state operations. The state consists of:

```
modes and flags word:

8000 -- unused

6000 -- rounding direction:

0000 -- to nearest

2000 -- toward +INF

4000 -- toward -INF

6000 -- toward zero (chop)

1F00 -- error flags, from high to low order:

1000 -- inexact result

0800 -- division by zero

0400 -- floating overflow

0200 -- floating underflow

0100 -- invalid operation
```

```
0000 -- not rounded up in magnitude
0080 -- rounded up in magnitude

0060 -- precision control:
0000 -- extended
0020 -- double
0040 -- single
0060 -- ILLEGAL

001F -- halt enables, correspond to error flags
```

halt vector:

32-bit address of alternate exit from package

11. Halts

When an error arises for which the corresponding halt is enabled, a trap is taken through the vector in the floating-point state area. The halt routine is called as a Pascal procedure of the form

The only way to return to the package from a halt is to initiate a new floating-point operation. There is no way to resume execution of the halted operation.

The state-related operations never halt. The binary-decimal conversions do not halt, though the individual operations they employ (such as multiplication to form 10^N for some integer N) might halt.

12. Other Pseudo-Machines

The package is simple and general enough to be the basis for pseudo-machines with register architectures like the 68881 or the Z8070 or with an evaluation stack like the Intel 8087. What is needed is simply the mechanism to compute addresses in the register file or stack (and check for internal consistency), and the set of functions required to manipulate that isolated data file (e.g. duplicate the top stack element, negate a register).

13. Arithmetic Abuse

The package is designed to be as robust as possible but it is not bullet-proof, since it is specified to modify the stack. If the user passes

illegal addresses, a memory fault may arise when the package attempts to access the operands. And if the user passes the wrong number of address operands, then in general the stack will be irreparably damaged. Operation is undefined if ILLEGAL values are used in the opword parameter.

14. Size and Performance

FP68K is about 4000 bytes long. On a 4mhz system it executes the simplest arithmetic operations in about 0.4ms and requires just over 1.0ms for a full extended multiply. Divide and square root are longer yet.

Comparative timings show that, for double format operations, FP68K is just faster than the AMD 9512 on Lisa and is about twice twice as fast as the Motorola 68341 code. For single format operations, FP68K is about half as fast as the Lisa single-only package, which is just slower than the 9512.

15. Floating-Point at a Glance

Figure 1 at the end of this document illustrates the basic control of flow in the execution of the floating-point package. The figure is followed by a list of observations on the behavior of the package, and of IEEE arithmetic in general.

- 1. The package has a single entry point.
- 2. The package has two exit points, one for normal subroutine returns and one for halts through a vector.
- 3. Three classes of operations are distinguished: arithmetic operations, binary-decimal conversions, and accesses to the state word and halt vector.
- 4. The not-a-number symbols, NANs, are detected at the start of each operation. Of them, signaling NANs are the most virulent; they always trigger the invalid operation exception. Quiet NANs propagate through operations; a precedence rule determines which is output if two are input.
- 5. Invalid operations always result in a quiet NAN output. In the case of the discrete types INT16, INT32, COMP64, the output value is all zero bits except for a leading one bit (that is, 100000...). Floating-point NANs contain an error code to indicate their origin (such as 01 for square root of a negative number).
- 6. When the input operands are unpacked, the special cases 0, FNZ (finite nonzero number), and INF (infinity) are detected. This expedites special cases such as

- 7. When 0 or INF results from a trivial operation like the example above, no further processing is required before the value is packed. All nontrivial floating-point results are subject to precision and range coercion to assure that they fit in the intended destination.
- 8. Integer results are subject to coercion to detect overflow.
- 9. Floating-point NAN results are coerced by chopping them to the precision of the destination, and checking that a legitimate value results.
- 10. Comparisons require special care, since they produce no results but rather modify the CPU condition-code register. Comparisons, even when NANs are involved, must bypass the coercion steps.

This is a brief guide to the program FP68K, a software implementation of proposed IEEE standard P754 (Draft 10.0) for binary floating-point arithmetic. This guide is intended to aid a programmer wishing to understand the workings of FP68K.

The code

The software is in the assembly language of the Motorola MC68000, following the Apple "TLA" syntax of the Lisa assembler. FP68K is non-self-modifying, position-independent code. It has no local data area, that is it uses dynamically allocated stack area for all of its temporaries. FP68K is one large subroutine whose single entry point has the name FP68K.

The code is separated into the functionally distinct files:

```
FPDRIVER.TEXT -- "includes" the other files...
             -- defines set of named constants
FPEQUS. TEXT
FPCONTROL.TEXT -- organizes the flow of control
FPUNPACK.TEXT -- unpack input operands to intermediate format
              -- add and subtract
FPADD. TEXT
              -- multiply
FPMUL.TEXT
FPDIV.TEXT
              -- divide
FPREM. TEXT
              -- remainder
FPCMP.TEXT
              -- compare
FPSQRT.TEXT
              -- square root
              -- floating <--> floating,integer conversions
FPCVT.TEXT
FPSLOG.TEXT
              -- logb, scalb, and class appendix functions
              -- handle "Not A Number" symbols
FPNANS.TEXT
FPCOERCE.TEXT -- post-normalize, round, check over/underflow...
              -- pack result to storage format
FPPACK.TEXT
              -- non-arithmetic operations
FPODDS.TEXT
              -- decimal --> binary conversion
FBD2B.TEXT
FBB2D.TEXT
              -- binary --> decimal conversion
FBPTEN.TEXT -- computes 10 N for nonnegative integer N
```

As noted, FPDRIVER.TEXT is a short file which simply includes the other files between the ".PROC" header and ".END" trailer.

Assembling FP68K

Assemble the file FPDRIVER.OBJ to produce the FP68K object file.

The one system dependency of FP68K is its access of the floating-point state area, as discussed in the "System Implementor's Guide". Near the top of FPCONTROL.TEXT is the code which pulls the address of the the 3-word state area into register AO. This code will typically require modification when FP68K is moved to a new system. The well-marked comment within FPCONTROL.TEXT indicates the different access schemes systems might use. If the state area

is to be located using a constant defined in a public "include" file, then that file should be included within FPDRIVER. TEXT. See the comment there for details.

Other than its access to the state area, FP68K is intended to system-independent and should not be tailored recklessly.

Control flow

There are three fundamentally distinct classes of operations performed by FP68K: basic arithmetic, binary-decimal conversions, and manipulations of the floating-point state area. The last of these, namely reading and writing the state word and the halt vector, is trivial and needs no explanation beyond the simple code contained in FPODDS.TEXT.

The basic arithmetic operations are illustrated in the flow chart at the end of this note. The chart is marked to distinguish the function of the various files listed above.

The binary-decimal conversions are quite different from the basic operations, and are not described by the basic flow chart. The conversions might better be thought of as subroutines which have been implemented within FP68K as a matter of architectural convenience. The conversions invoke FP68K itself to perform various basic operations like multiply and divide. The binary-decimal algorithms are described in considerable detail in the attached paper "Accurate, Yet Economical Binary-Decimal Conversions" by J. Coonen.

Exponent calculations

FP68K manipulates exponents in a way that might seem surprising at first glance. The P754 extended format, on which all FP68K arithmetic is based, has a 1-bit sign, 15-bit exponent, and a 64-bit significand. However, the actual exponent range is not 0 to 32767 (biased by 16383) as the 15-bit exponent field would suggest. Rather, it is -63 to 32767 because of the presence of tiny denormalized numbers; this is "just a little bit" beyond the stated 15-bit range. (See the attached paper "Underflow and the Denormalized Numbers" by J. Coonen for a discussion of tiny values in P754 arithmetic.)

Because the operations multiply and divide require the addition and subtraction, respectively, of operand exponents in forming their intermediate results, the implementor typically expects to have one extra exponent bit for intermediate calculations. Thus for P754 extended format calculations, there is need for "just a little bit" beyond 16 exponent bits. This elusive 17-th bit is discussed in yet another attached paper, "Are 17 Exponent Bits Too Many?" It is shown there that 16 bits suffice, if care is taken to perform some extra tests in the right places.

On the 68000 it turns out to be convenient to perform exponent calculations in the ADDRESS REGISTERS — with a full 32 bits. The address registers provide just the right functionality: add, subtract, and compare. And since floating-point arithmetic is computation-intensive on a small data set, only a few of the address registers are actually needed for addresses.

Finally, 16-bit constants like the exponent bias may be added into the 32-bit exponents with a 2-word instruction, since for "address" calculations the constant is first sign-extended out to a full 32 bits.

Bit field encodings

This section describes the various bit fields used by FP68K. Some of them, like the opcode and the state word, are visible to programs invoking FP68K. Others, like the rounding and sign bits, are local to FP68K.

The OPCODE is the last word pushed on the stack before calling FP68K. It is composed of the fields:

```
3800 -- "non-extended" operand format:
     0000 - x80
     0800 -- f64
     1000 -- f32
     1800 -- ILLEGAL
     2000 -- intl6
     2800 -- int32
     3000 -- comp64
     3800 -- ILLEGAL
07E0 -- must be zero
001F -- operation code:
     0000 -- add
     0002 -- subtract
     0004 -- multiply
     0006 -- divide
     0008 -- compare
     000A -- compare (invalid if UNORDERED)
     000C - remainder
     000E -- x80, f64, f32, int16, int32, comp64 --> x80
     0010 -- x80 --> x80, fo4, f32, int16, int32, comp64
     0012 -- square root (in x80)
     0014 - round to integer (in x80)
     0016 -- truncate to integer (in x80)
     0018 -- scale by unbiased power of 2
     001A -- replace by unbiased exponent
     001C -- classify the floating input
     001E -- ILLEGAL
     0001 -- put state word
     0003 -- get state word
     0005 -- put halt vector
     0007 -- get halt vector
     0009 -- decimal --> floating convert
     000B -- floating --> decimal convert
     000D -- negate
     000F -- absolute value
     0011 -- copy sign
     0013 -- nextafter
```

```
0015 -- set exception

0017 -- procedure entry protocol

0019 -- procedure exit protocol

001B -- test exception

001D and 001F are ILLEGAL
```

The STATE word is static data that perseveres across calls to FP68K. As such, it must live in an area outside FP68K, defined by the host system. Typically the state word (and the halt vector, which is a 32-bit address) will live in the system's "per-process data area", perhaps a fixed location in memory or a fixed offset from some reserved address register. Although the STATE word is directly available to the programmer, typical access will be through an intermediate layer of software (available, say, in a Pascal unit) that insulates the programmer from the details of the actual bit encodings. The STATE word is composed of the fields:

```
8000 -- unused
6000 -- rounding mode:
     0000 -- to nearest
     2000 -- toward +INF
     4000 -- toward -INF
     6000 -- toward 0 (chop)
1F00 -- error flags:
     1000 -- inexact result
     0800 -- division by zero
     0400 -- floating overflow
     0200 -- floating underflow
     0100 -- invalid operation
0080 -- rounding of last result
     0000 -- not rounded up in magnitude
     0080 -- rounded up in magnitude
0060 -- precision control:
     0000 -- extended
     0020 -- double
     0040 -- single
     0060 -- ILLEGAL
001F -- exception halt enables:
     (correspond to error flags above)
```

After preliminary decoding in FPCONTROL. TEXT, the OPCODE is expanded out into the following 16-bit form:

```
8000 -- nonzero iff result has single precision and range
4000 -- nonzero iff result has double precision and range
3800 -- source operand format:
```

(same encoding as in OPCODE)

0700 -- destination operand format: (same encoding as in OPCODE)

0080 -- nonzero iff destination operand is input

0040 -- nonzero iff source operand is input

0020 -- nonzero iff destination operand is output

001E -- operation code:
(same encoding as in OPCODE but with low bit 0)

0001 -- nonzero iff two-address operation

The ROUND BITS, known as "guard", "round", and "sticky" in documentation about P754, are kept in a 16-bit word. Roughly speaking, the guard and round bits are the two bits beyond the least significant bit of the intermediate result, and the sticky bit is the logical Or of all bits thereafter. The sticky bit is necessary to implement the rounding modes of P754. The ROUND BITS are kept as:

8000 -- guard bit

4000 -- round bit

3F00 -- 6 extra round bits

00FF -- sticky bits

The reason for keeping an entire byte of sticky bits lies in the 68000 instruction set. The archetype operation involving the sticky bit is the right-shift. Any time a bit is shifted off the low end of the sticky "byte", it must be logically Or-ed back into sticky. This is done with the 68000 "SCS" instruction, which sets a given byte to all 1s if the carry bit is set, and clears the byte to 0 otherwise. Typically, a bit is shifted off to the right, it is SCS-ed into an auxiliary byte, and that byte is Or-ed into the sticky byte. Although this is the typical use of the sticky byte, the programmer should not assume that the sticky byte is always either all Os or all 1s. Sometimes, such as in the right shift after a carry-out in ADD/SUB, the logical Or will be omitted since it is known that if a 1 was shifted out of the sticky byte there will necessarily be another 1 left in sticky.

The operands' SIGNS are kept together in a byte as follows:

80 -- source operand sign

40 -- destination operand sign

20 -- Exclusive Or of the two operands' signs

1F -- unused, but not necessarily zero

If there is just one input operand, its sign is in the high order bit. The Exclusive Or is computed just once, at the start of every arithmetic operation. Not only is it required for many common operations (+, -, *, /, REM, CMP), but it is costly in time and space because of the inefficacy of the

68000 bit instructions, so it is worthwhile to implement the code sequence just once.

The CCR (condition code register) bits of the 68000 are modified by every arithmetic operation, though only the compare instructions leave them in a well defined state. A CCR word is maintained by FP68K:

FFE0 -- unused, forced to 0 0010 -- X = Extend 0008 -- N = Negative 0004 -- Z = Zero 0002 -- V = Overflow 0001 -- C = Carry

The compare operations encode their results as follows:

RELATION	FLAGS:	X	N	Z	V	С
LESS		1	1	0	0	1
EQUAL		0	0	1	0	0
GREATER		0	0	0	0	0
UNORDERED		0	0	0	1	0

See the FP68K programmer's manual for the software applications of the CCR field.

Register usage

The key to the speed (such as it is) and compactness of FP68K is that its entire working data set may be held in the 68000 register file. Immediately upon entry, FP68K saves registers D0-D7, A0-A4 on the stack. Then the registers are loaded up as the operation proceeds. Several of the registers have a meaning that perseveres across nearly the entire instruction. The following list gives a rough idea of register usage:

D7 hi -- CCR word
D7 lo -- round bits
D6 hi -- opcode word
D6 lo -- error byte (hi) and sign byte (lo)
D5 -- low 32 source (later result) significant bits
D4 -- high 32 source (later result) significant bits
D3-D0 -- scratch area

A

```
7
      -- SP = stack pointer
      -- stack link pointer
A6
      -- Mac globals poionter
A5
      -- source (later result) exponent
A4
      -- destination exponent
A3
      -- low 32 destination significant bits
A2
      -- high 32 destination significant bits
A1
      -- pointer to 3-word state area
A0
```

1 November 83

Draft 1.6

Mac FP Software Program Notes

62

Of course, the arithmetic operations may be viewed as transformations of the register file. Following this view, a set of register maps are included at the end of this note. They are keyed to MILESTONES marked in the source code. The maps indicate register dependencies, and as such should aid in any modification of FP68K. Some maps simply indicate the state of the register file at a given point, and some indicate register use in a routine, such as the widely used right-shift procedure RTSHIFT.

For convenience the maps are printed on onion skin paper; a reference sheet slips under the map to fill in the register mask.

Register DO is modified by the REMAINDER operation, in which case a partial integer quotient is returned in $DO \cdot W$.

Stack usage

When called, FP68K assumes that the stack has the form:

```
ADDRESS 3 -- used for decimal format code only
ADDRESS 2 -- source pointer, if any
ADDRESS 1 -- destination pointer
OPCODE -- one word
RETURN ADDRESS
```

The number of address operands depends on the operation. FP68K then allocates 3 more stack words:

COUNT -- number of bytes in original call frame HALT ADDRESS

This frame is used if a halt is taken. The COUNT field allows the halt handler to simply pop the original operands and return, if desired.

Above this frame, FP68K pushes registers D0-7, A0-6. In the progress of an operation, up to 6 more words of stack may be used. The total stack usage, after the call, is then up to 3+32+6=41 words. The binary-decimal conversions may use twice this much since they invoke FP68K to perform basic arithmetic operations.

Conditional assembly

the floating-point state area is loaded into register AO at the start of FPCONTROL.TEXT. Since the location of this area is system-dependent, conditional assembly is used to locate the field. Of course, this means that the effective address of the state area must be known at assembly time.

Conditional assembly is also used to resolve syntactic inconsistencies between various 68000 assembly language formats. The program counter (PC) relative addressing modes are heavily used in the implementation of jump offset tables within FP68K. A typical use is the instruction sequence:

1 November 83

Draft 1.6

Mac FP Software Program Notes

63

MOVE.W JMPTAB(DO),DO JMP JMPTOP(DO)

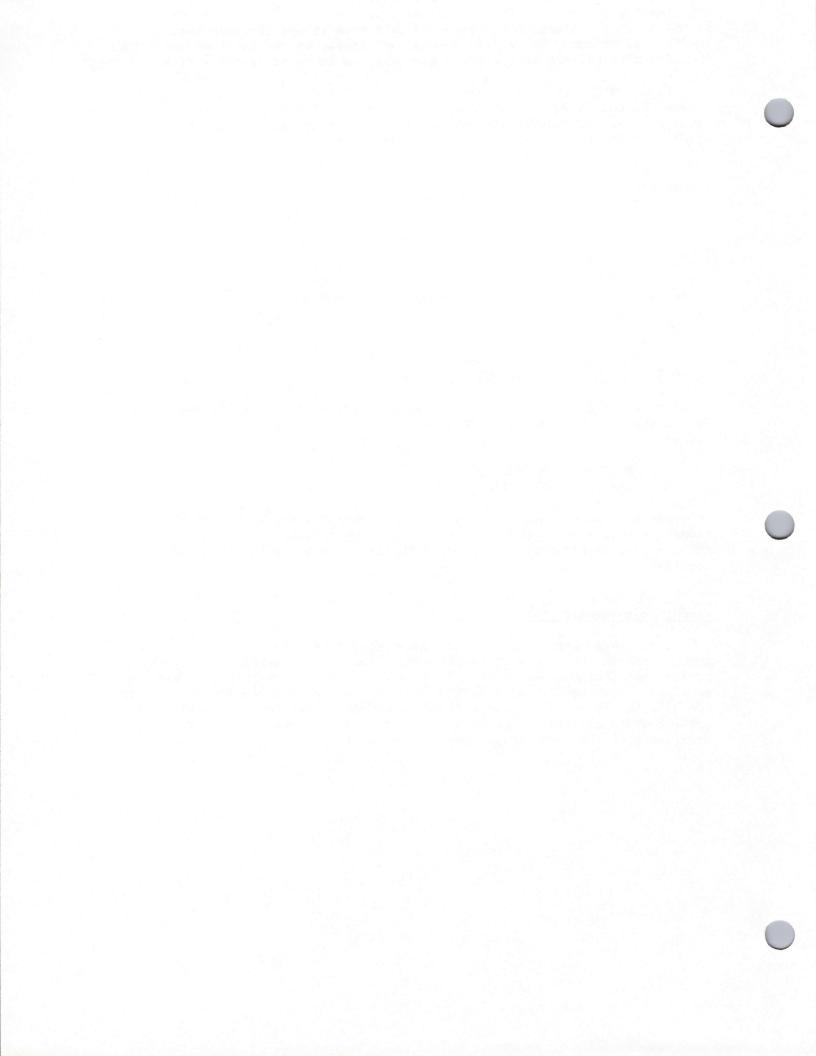
Here JMPTAB is a table of address offsets from the label JMPTOP, and register DO contains a word index into JMPTAB. Some assemblers force the programmer to write:

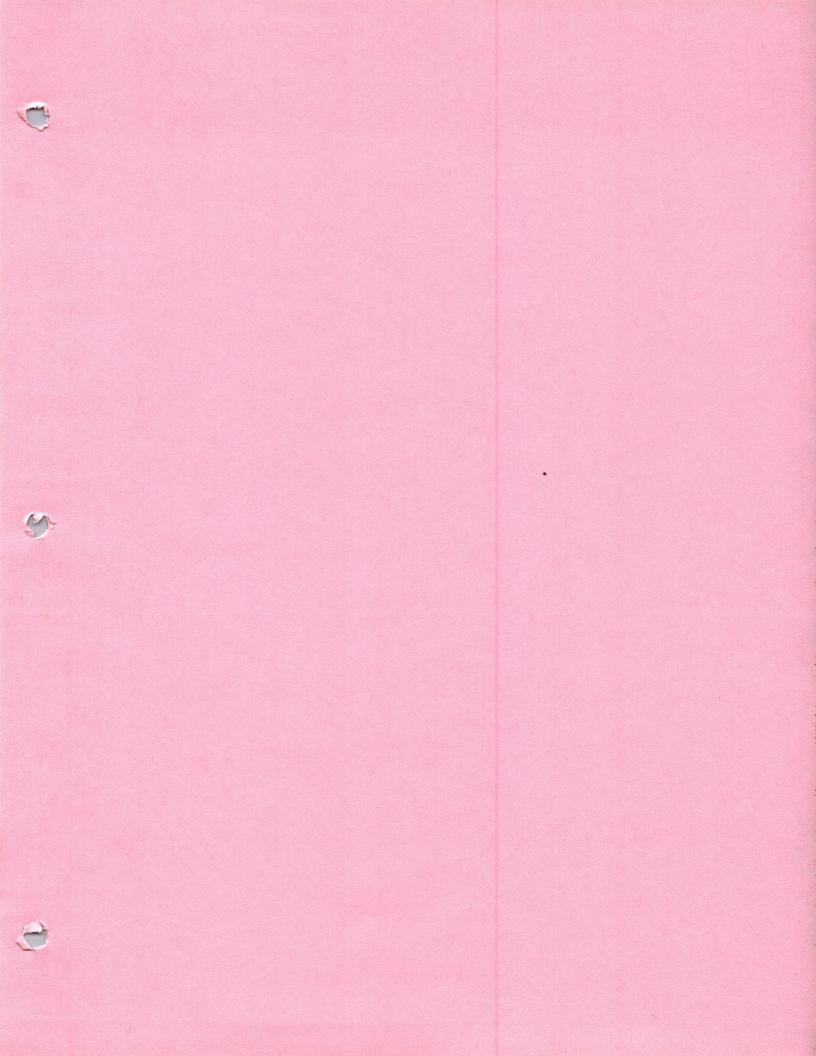
MOVE.W JMPTAB(PC,DO),DO JMP JMPTOP(PC,DO)

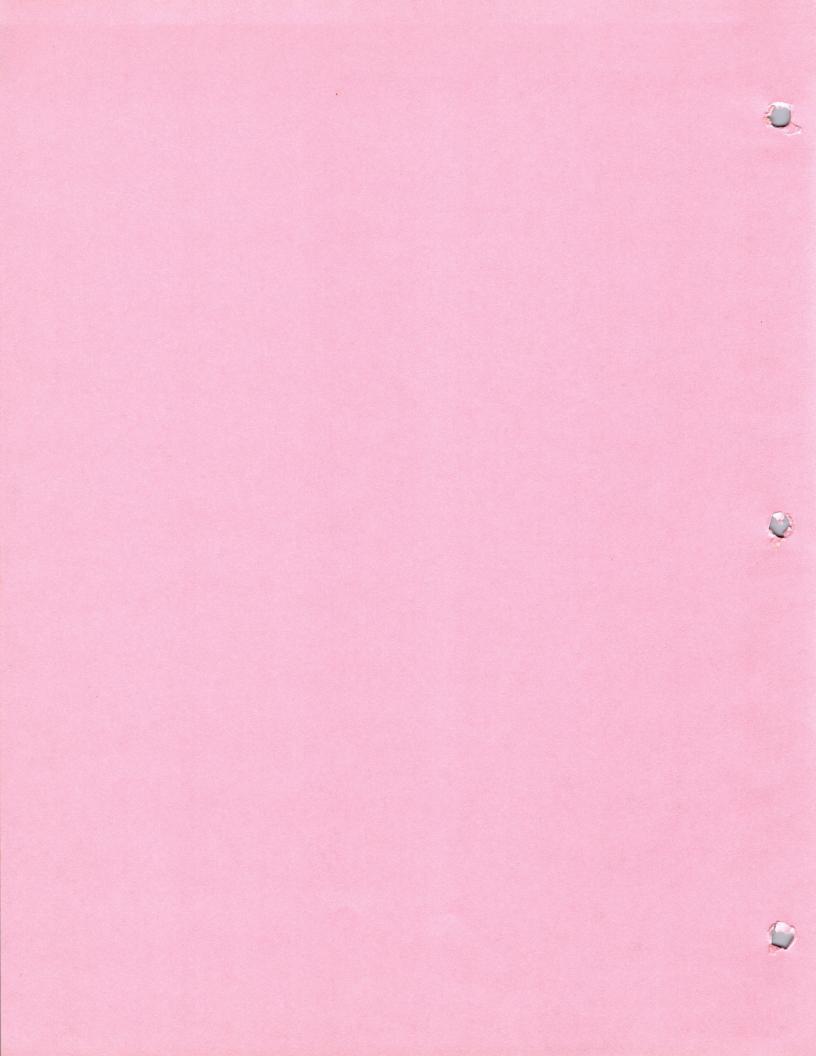
in order to assure PC-relative addressing. However, the Lisa assembler PROHIBITS this syntax, although it produces the desired code. An assembly flag is used to generate whichever of the two formats is suitable for a given compiler.

Pascal enumerated types

Lisa Pascal attempts to encode enumerated types in byte fields, which are then stored as the high byte of the target word. This affects structures like DecForm and Decimal, defined in the Pascal interface (see that document for details). Although the most seriously affected programs are the test drivers, the affected files in the basic package are FBB2D.TEXT and FBD2B.TEXT. Those files contain explicit comments when a byte test is used where an Apple III programmer (for example) might expect a word test.







cal Interface to

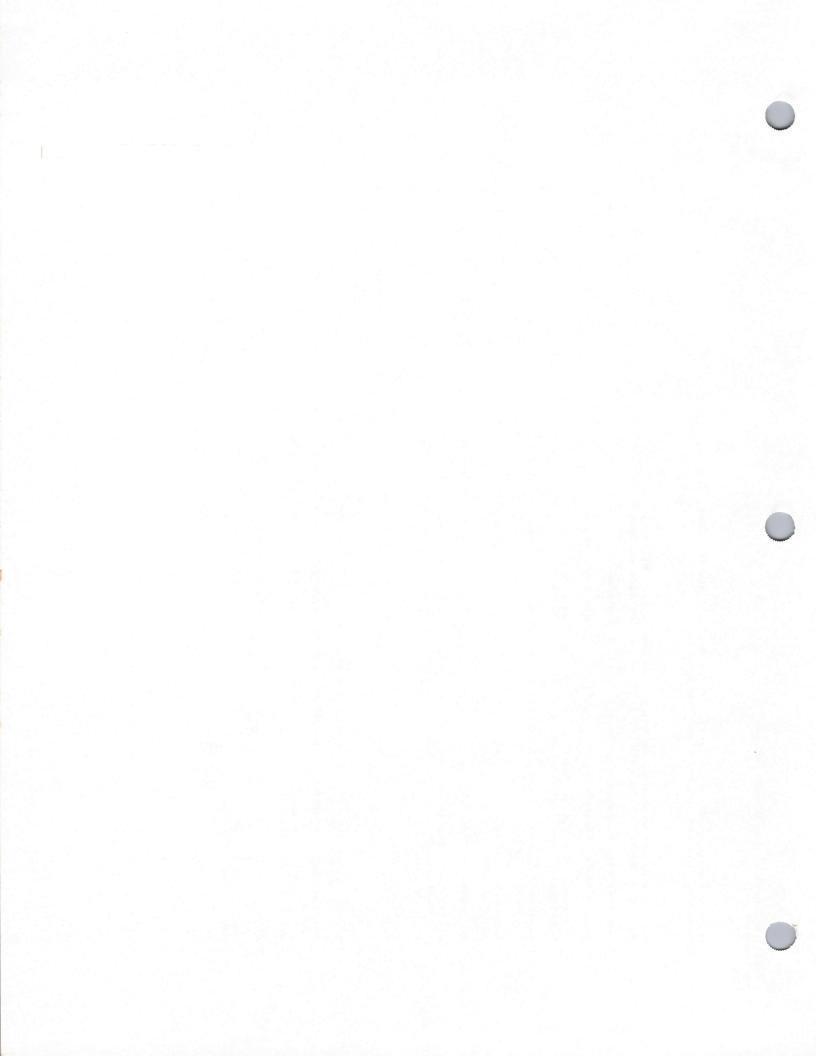
Pascal Interface to the Memory Manager 25 Sep 83 LAK The memory manager Pascal interface files MemTraps and HeapZone have been updated and changed. The differences are:

eauZone

- memory manager error equates changed to MemFullErr, NilHandleErr, MemWZErr, MemPurErr, and NoErr; this is now the same as SysErr.
- * the zone record has been updated to reflect the ROM.
- * HeapValidate, LastHandle, and LastPtr calls have been removed.
- Resrymem, SetApplBase, and BlockMove procedures added. GZCritical and GZSaveHnd added for use of GrowZone functions.
- + LastError function is now MemError and it returns an integer error code (like ResError).
- DisposeHandle and DisposePtr are now DisposHandle and DisposPtr to agree with SysMacs.
- Purgemen and CompactMem no longer return pointers to free memory blocks.
- MaxSize constant is now defined; this should be passed to CompactMem and PurgeMem to make them compact and purge the entire heap (which is usually desired).
- * InitZone parameter order has been changed.

MemTraps -

- * Changed to reflect the above HeapZone changes.
- * InitZone and SystemZone calls did not previously work; hopefully they do now.
- * All calls except MemError, GZCritical, and GZSaveHnd store an error code which may be retrieved by MemError.
- Error codes for GetHandleSize, GetPtrSize, and RecoverHandle now are set correctly. If GetHandleSize or GetPtrSize encounter errors, they now return a size of 0; the correct error code is recorded for MemError. RecoverHandle unconditionally sets error code 0 as the OS fails to



```
Unit Decone;
```

```
Unsigned byte for fontmgr }
blind pointer }
pointer to a master pointer }
32 bit offset from a zone to a master ptr }
32 bit pointer to heap zone }
                                                                                                                                              trying to purge a locked or non-purgable block
                                                                                                  Not enough room in heap zone }
Handle was NIL in HandleZone or other }
WhichZone failed (applied to free block)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        pointer to the start of a heap zone size of a block in bytes }
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  ( reserved for future )
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   pointer to a relocatable block
                                                                                                                                                                                       ( Max data block size is 512K bytes
Copyright 1983 by Apple Computer, Inc. All Rights Reserved.
                                                                                                                                                                                                                                                                                                                     pointer to a procedure )
                                                                                                                                                                                                                                 any byte in memory )
                             Last changed 26 September 1983
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     error code )
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         ProcPtr;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     : Ptr);
                                                                                                                                                                                                                                                                                                                                                                                          Longint;
                                                                                                                                                                                                                                                                                                                                                                                                        Procetr;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                               Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     Longint;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    Procetr;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                       Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                    Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                                  Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                Integer;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      CompactMem(cbNeeded: Size); Size;
                                                                                                                                                                                                                                                                                                                                                              Ptr;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  Ptr;
                                                                                                                                                                                                                                                                                                                                                                             Ptr;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              Procedure SetApplBase(startPtr: Ptr);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     limitPtr, startPtr
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       CntHandles:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   MinCBFree:
PurgeProc:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    Procedure PurgeMem(cbNeeded: Size);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               Procedure ResryMem(cbNeeded: Size);
                                                                                                                                                                                                                                                                                                                                                                                                                   MoreMast:
Flags:
CntRel:
                                                                                                                                                                                                                                            0..255;
^SignedByte;
                                                                                                                                                                                                                                                                                                                                                                         HFstFree:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              SparePtr:
                                                                                                                                                                                                                                                                                                                                                              PurgePtr:
                                                                                                                                                                                                                                                                                                                                                                                         ZCBFree:
GZProc:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         Cnt Empty:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              AllocPtr:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              неарData:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                             CntNRel:
                                                                                                                                                                                                                             -128..127;
                                                                                                                                                                                                                                                                                                                                                                                                                                                             MaxRel:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         moreMasters:
                                                                                                                                                                                                                                                                        APtr;
LongInt;
                                                                                                                                                                                                                                                                                                                                                BKL im:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     Langint;
Integer;
                                                                                                                                                                                    : $800000;
                                                                                               MemfullErr = -108;
NilHandleErr = -109;
MemWZErr = -111;
MemPurErr = -112;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      Procedure InitZone(growProc:
                                                                                                                                                                                                                                                                                                                                  Record
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          'auozv
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               Procedure SetZone(hz: THz);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   FreeMem: LongInt;
                                                                                                                                                                                                                                                                                                   Ptr;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            ApplicZone: THZ;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           SystemZone: THZ;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              Ptr:
                                                                                                                                                                                                                                                                                                                    Ptr;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           Procedure InitApplZone;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   GetZone: THZ;
                                                                                                                                                                                                                                                                                    11 11
                                                                                                                                                                                                                              15
                                                                                                                                                                                                                                                            11
                                                                                                                                                                                                                                                                         18
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    15
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  IF 11
                                                                                                                                                                                                                                 SignedByte
                                                                                                                                                                                                                                                                                                    NRe 1 Hand 1 e
                                                                                                                                                                                                                                                                                       RelHandle
                                                                                                                                                                                      MaxSize
                                                                                                                                                                                                                                                                                                                    Procetr
                                                                                                                                                                                                                                                                        Handle
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  Memerr
RelPtr
                                                                                                                                                            NOErr
                                                                                                                                                                                                                                              Byte
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     Size
                                                         Interface
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     Function
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        Function
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              Function
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            Funct ion
                                                                                    Const
                                                                                                                                                                                                                  lype
```

Function MaxMem(Var grow: Size): Size; Function TopMem: Ptr;

procedure SetGrowZone(growZone: ProcPtr); Procedure SetApplLimit(zoneLimit: Ptr); Function NewPtr(byteCount: Size); Ptr; Procedure DisposPtr(p: Ptr); Function GetPtrSize(p: Ptr); Size; Procedure SetPtrSize(p: Ptr; newSize; Size); Function PtrZone(p: Ptr); THz;

Function NewHandle(byteCount: Size): Handle;
Procedure DisposHandle(h: Handle);
Function GetHandleSize(h: Handle): Size;
Procedure SetHandleSize(h: Handle): Size;
Function HandleZone(h: Handle): THz;
Function RecoverHandle(rp: RelPtr): Handle;
Procedure EmptyHandle(h: Handle);
Procedure ReAltyHandle(h: Handle);

Procedure HLock(h: Handle); Procedure HUnLock(h: Handle); Prucedure HPurge(h: Handle); Procedure HNoPurge(h: Handle); Procedure BlockMove(srcPtr, destPtr; Ptr; byteCount: Size); Function MemError: MemErr;

Function GZCritical: Boolean; Function GZSaveHnd: Handle;

Implementation

(no code here, it's all in MemTraps.text)

End.

GrowZone function in graphic detail as input to pub's document on the memory manager. The analysis may be of interest but the conclusions are what is Inis document is intended to describe the memory manager's calls to a zone's important (found at the end of this document).

The following memory manager calls may invoke the GrowZone function:

SetPtrSize NewHandle SetHandleSize ReallocHandle ResrvMem

zone's GrowZone function; CompactMem will compact only and PurgeMem will purge only. These 3 calls plus the above 6 calls are the only memory manager calls which ever cause heap movement; dereferenced handles only become invalid after these calls or other system calls which call these directly or MaxMem will purge blocks but will never try to grow the zone or invoke the indirectly. The GrowZone calls may be differentiated into 9 cases (this section is included for my reference only - it refers to internal memory manager routines). Three low-memory variables used by the memory manager, GZRootHnd, GZRootPtr, and GZMoveHnd differentiate these cases somewhat:

- 1) ResrvMem > MakePtrSpc > MakeSpace > SafeReloc > RelocRel > AllocBk > BkCompactS GZRootHnd = GZRootPtr = 0, GZMoveHnd # 0 (non-crit, move blk out of way)
- 2) NewPtr > AllocBk > BkCompactS

G2RootHandle = G2RootPtr = G2MoveHnd = 0 (crit, need spc for nrel blk)

3) SetPtrSize > SetSize > MakeSpace > SafeReloc > RelocRel > AllocBk > BkCompactS GZRootPtr # 0, GZMoveHnd # 0 (non-crit, move blk out of way) GZRootHnd = 0.

(non-critical because Memory Manager never guarantees it can grow a pointer)

- 4) NewHandle > NextMaster > HMakeMore > MakePtrSpc > MakeSpace > SafeReloc > RelocRel GZRootHnd = GZRootPtr = 0, GZMoveHnd # 0 (non-crit, move blk out of way) AllocBk > BkCompactS
- 5) NewHandle > NextMaster > HMakeMore > AllocBk > BkCompactS

GZRootHnd = GZRootPtr = GZMoveHnd = 0 (crit, need spc for master ptr blk)

6) NewHandle > AllocBk > BkCompactS

GZRootHnd = GZRootPtr = GZMoveHnd = 0 (crit, need spc for rel blk)

- 7) SetHandleSize > SetSize > MakeSpace > SafeReloc > RelocRel > AllocBk > BkCompactS of way) GZRootPtr ♯ 0, 0 # GZRootHnd # GZMoveHnd # 0 (non-crit, move blk out
- 8) SetHandleSize > SetSize > RelocRel > AllocBk > BkCompactS

GZRoutPtr ¥ 0, GZRootHandle = GZMoveHnd # 0 (crit, need spc for rel blk)

9) ReAllocHandle > AllocBk > BkCompactS

GZRootHnd # 0, GZRootPtr = GZMoveHnd = 0 (crit, need spc for rel blk)

Or these 9 cases, 5 are critical (the memory manager needs the memory for a block).:
When non-critical cases occur, the memory is probably starting to get full (or
an unusually large block is involved). The first three non-critical cases
are more important than the last since two involve allocation of non-movable
blocks as low in memory as possible and the third will cause SetPtrSize to
fail if memory is not found. It is still ok to punt in any non-critical case and most applications will. GZRootHnd = GZRootPtr = GZMoveHnd = O (crit, need spc for nrel blk)
GZRootHnd = GZRootPtr = GZMoveHnd = O (crit, need spc for master ptr blk)
GZRootHnd = GZRootPtr = GZMoveHnd = O (crit, need spc for rel blk)
GZRootPtr # O, GZRootHnd = GZMoveHnd # O (crit, need spc for rel blk)
GZRootPtr # O, GZRootPtr = GZMoveHnd # O (crit, need spc for rel blk) NewHandle 2) NewPtr 5) NewHand 6) NewHand 8) SetHand 9) ReAlloc critical:

- GZRootHnd # - GZRootHnd = NewHandle SetHandle

Resrvmem - GZRootHnd = GZRootPtr = 0, GZMoveHnd # 0 (non-crit, move blk out of way)
SetPtrSiz - GZRootHnd = 0, GZRootPtr # 0, GZMoveHnd # 0 (non-crit, move blk out of way)
NewHandle - GZRootHnd = GZRootPtr = 0, GZMoveHnd # 0 (non-crit, move blk out of way)
SetHandle - GZRootPtr # 0, 0 # GZRootHnd # GZMoveHnd # 0 (non-crit, move blk out of way) ReallocHa

It may been seen that a growzone call is critical, i.e., the growzone function must return memory if either:

GZRootHnd = GZMoveHnd, or GZMoveHnd =

The following code may be used to differentiate the two:

GZRootHnd, DO GZMoveHnd, DO NonCrit Move. L BEq. S Cmp. L MyGZFunc

; try to free up some memory but don't dispose the blocks ; pointed to by GZRootHnd, GZMoveHnd, or GZRootPtr CritCase

; returning none

C16.L RIS Memtraps now provides a function G2Critical which returns true when the case is critical.

The main reason for the three low memory variables is let the application easily

tell which handles or pointers may be disposed or purged. If the call resulting in the memory manager invocation of the GrowZone function was a SetHandleSize or ReallocHandle, it would be disasterous for the GrowZone function to dispose the handle which was being made longer or reallocated; this handle is GZRootHnd. If the original call was SetPtrSize, GZRootPtr is the original pointer which should not be disposed. Finally, GZMoveHnd points to a block being moved to room to satisfy the original call; this block may be purged by the GrowZone function but should not be disposed.

If the above code is used to skip non-critical cases, only GZRootHnd is important and the GrowZone function need only pay heed not to deallocate, purge, or change the size of that handle.

MemTraps now provides a function G2SaveHnd which returns the value of G2RootHnd.



MAC-PRINT

TO: MacPrint Users DATE: November 5, 1983

SLBJECT: The .Print Driver FROM: Owen Densmore

Introduction & Installation

The Print driver lives in the System.rsrc file as ".Print" and ResID=2, RefNum=\$FFFD. [Also in System.rsrc is a string (ResID=\$E000, -8192) containing the file name of the Print resource file for the higher level print code. This is currently "ImageWriter". There may be more stuff later, too.]

The driver currently has two source files: PrDrvr68k and CiDrvr68k which are linked to PrDrvr.obj. This is stored in the Print resource file under an alias: .XPrint, id=\$E000. This is done as a backup for the driver and to allow installation of the driver into System.rsrc. Until the installation program is written, we also store the driver, and any other System.rsrc printing resources in a "Stub" resource: PrSys.rsrc. To install these into System.rsrc, simply put PrSys.rsrc on your Mac disk and use RMover:

Open PrSys
Copy PrDryr, the Str. ... in

Copy PrDrvr, the Str, ... into the scrap

Close Prsys

Open System.rsrc (without selecting anything!)

Paste

Quit

[..You may have to move journal to some other ID if it is at id 2. Just move to a free orn id.]

Driver calls

The driver contains the following calls:

Status: Font Mgr dev info call.

Controls:

iPrBitsCtl = 4;

iPrIOCtl = 5;

iPrEvtCt1 = 6;

iFMgrCtl = 8;

The FMgr control call (8) is the "tail hook" which allows the printer to override the font manager's font selection.

The Print Event Control (6) uses one long param for handling two special Bitmap printing cases:

Msg Format: Byte 3=0; Byte 2=\$FF for Screen, \$FE for Top Folder; Byte 1 & 0 = Print Driver's RefN CSParam = The KeyM[ap] event.message generated by special key combinations:

1PrEvtAl1 = \$00FFFFFD;

1PrEvtTop = \$00FEFFFD;

The FMgr control call (8) is the "tail hook" which allows the printer to override the font manager's font selection.

The Printer IO Control (5) uses three long params for writing raw data to the printer:

CSParam = pointer to a byte string

CSPeram+4 = a long count of bytes

SParam+8 = pointer to an idle proc; use NIL for none.

The Bitmap Printing Control (4) uses three long params for printing a portion of a bitmap:

CSParam = pointer to QuickDraw bitmap

CSParam+4 = pointer to rectangle within bitmap [local coords]
CSParam+8 = a control parameter: use 0 for screen printing

Thus to print the entire screen, use params: screenBits, screenBits.bounds, 0. To print the contents of a window, use params: window.portBits, window.portRect, 0.

PrScreen, some driver glue

The unit PrScreen [<100 bytes] contains aids to use of the Driver:

PROCEDURE PrDrvrClose:

PROCEDURE PrCtlCall (iWhichCtl: Integer; IParam1, IParam2, IParam3: LongInt);

A generalized Control proc for the Printer driver.

The main use is for bitmap printing:

PrCtlCall (iPrBitsCtl, pBitMap, pPortRect, lControl);

PROCEDURE PrBits (pBitMap: Ptr; --QuickDraw bitmap

Note PrCtiCall (iPrBitsCtl, @MyPort^.ScreenBits, @MyPort^.PortRect.Bounds,0) performs a screen dump of just my port's data.

Two special control calls are included in the driver for Screen printing from the key board:

PrCtlCall (iPrEvtCtl, IPrEvtAll, 0, 0); Prints the screen PrCtlCall (iPrEvtCtl, IPrEvtTop, 0, 0); Prints the top folder

These are handled by the system for key board access but can be called by anyone at any time. They can be very cheap printing for ornaments, for example!

Another useful call is used for sending raw data to the printer: PrCtlCall (iPrIOCtl, pBuf, lBufCount, pIdleProc);

These calls are available in two ways. One is simply to link PrScreen.obj to your program and use it as any other unit. See the program "Min" for sample use. The other way is for use by Apps using standard document (heavy!) printing. They simply call these through PrLink as they do for other printing procs. PrScreen is in its own segment, just as are the other printing segments.

System dependencies

The KeyM proc must be the latest version that generates the printer events. The Ram-based System must be the one that calls the Print driver from the Event patch. You need the RMaker that can use negative numbers. The Print driver must be installed in System.Rsrc

TO: MacPrint Users

DATE: November 7, 1983

SBECT: The MacPrint Interface

FROM: Owen Densmore

Introduction

This is a brief description of the new "Linkless" MacPrint.

Because MacPrint is is not part of the ROM code it must be included with the Application's code. Typically this would be done by including the MacPrint code in your Linker job. There are two reasons we don't do this. One is that we want to be able to configure new printers without re-linking the new print code into the Applications. The other is that we cannot assume that all OEM's are using the Lisa Workshop's Linker! MicroSoft, for example, runs an interpreted "C" environment with a VAX development system!

Our solution to this packaging problem is to provide four "PDEF" definition procs in a special printing resource file. These four def procs break the printing code into four disjoint code segments: Dialogs, Spooling, Draft printing, and Picture printing. In addition, there is a driver, ".Print", installed in System.rsrc which is accessed like any other driver with Open, Close, Status and Control calls.

A new printer is configured by supplying a new printer resource file, and installing its file name [ImageWriter, for example] and its .Print driver in System.rsrc.

Access to these "PDEF" procs is via a very small (374 byte) piece of Print Glue called "PrLink" which must be Linked (or some how included!) into your application. In addition, the .Print driver can be accessed directly. The driver and its use is discussed in detail in a seperate document.

The Pascal interface is "MacPrint.obj" and the Assembly interface is "PrEqu.text". There are four main groups of procedures: Init/Termination, Dialogs, Spooling/Oraft, and Picture Printing. This release is our "Beta" release. This means that the interface is as stable as we can make it. No further procedural interface changes will be made. Only changes internal to the code will be allowed.

Initialization

The Init/Termination code consists of:

PROCEDURE PrOpen; PROCEDURE Prolose;

These Open/Close the printing code by opening/closing two files: the current print resource file [ImageWriter.rsrc, for example] and the Print driver [.Print] which is part of the System.rsrc file.

Dialogs

The Dialogs maintain the primary printing data structure, the "Print" record. Note that one of these should be stored in each of your documents so that client use of printing can be "remembered". We'll discuss this protocol more later. The dialog procs are:

PROCEDURE PrintDefault (hPrint: THPrint);
This fills a handle to the defaulted Print record from the current print resource file. This does not actually dialog. It is used to initialize new "stationary" and to let "listing" style applications to get a valid Print record for printing without dialogs. The handle must be pre-allocated as 80 bytes.

FUNCTION PrStlDialog (hPrint: THPrint): Boolean;

The Mac Applications provide "visual fidelity", i.e. "What you see is what you get". This means that you must know something about the printing request before it is actually made! The PrStlDialog asks for the part of the print request that causes one print job to vary from another. For most printers, this is simply the page size and orientation for the document. No guarantee is made that this will always be so, however! The guarantee is only that enough information is obtained to fill out the part of the Print record called the Printo data structure, which will be described further below.

FUNCTION PrJobDialog (hPrint: THPrint): Boolean; The rest of the Print record is filled out by this dialog. It mainly consists of the page range and number of copies. For the Image Writer it also has the HiRes/LoRes/Draft choice and the type of paper feed.

The boolean result for both dialogs is the Dolt button: if true, the Client has clicked "OK". The Print record should be saved and, for the PrJob, the document spooled and/or printed. Note that the initial button settings are derived from the existing Print record values, and should be either an old, valid one, or a new, defaulted one.

Spooling and Draft Printing

The Spooling/Draft procs do one of two things: Spool a print file to disk, or provide for an Ascii like style of printing. Both are provided by setting up a graf port and intercepting QuickDraw calls by using our own versions of the QuickDraw bottleneck procs. Thus these embody the minimum use of printing by an App: you can either do "Cheap" ascii printing or spool the file to be printed later, "offline", by the Printer Application that will read and print the file. The interface is via four procs that "bracket" calls to QuickDraw:

FUNCTION Propendoc (hPrint: THPrint; pPrPort: TPPrPort; nTOBuf: Ptr): TPPrPort

pIOBuf: Ptr): TPPrPort;
Initialize the printing code for this document. The hPrint parameter is a handle to a valid Print record. The pPrPort is similar to the Window Manager's Storage parameter: if Non-NIL, I use it, rather than calling NewPtr. The IO Buffer pointer is passed allong to the OS: if NIL, it uses the volume buffer, otherwise it uses yours. The returned pointer is to the initialized Print "Port" which is simply a GrafPtr, its associated bottleneck procs, and a few extra longs for me. The code will initialize for Draft printing or for Pic file spooling by looking at the fDraft flag in the hPrint data. If the hPrint has a non-NIL IdleProc, it will be called by the draft printing proc.

PROCEDURE ProloseDoc (pPrPort: TPPrPort);
Puts the above stuff to bed: Flushes the Pic file directories or closes the print driver for draft printing.

PROCEDURE PrOpenPage (pPrPort: TPPrPort; pPageFrame: TPRect); Initializes the next page. The page frame rectangle is for wizards: set it to NIL.

PROCEDURE ProlosePage(pPrPort: TPPrPort);
Cleans up the Pic file data structures or ejects the current page.

See PrTest for sample usage:

pMyPort := PrOpenDoc (hPrint, pMyPort, pIOBuf);
FOR iPage := 1 TO iPages DO BEGIN { ..or WHILE NOT EOF(myDoc) DO BEGIN {
 PrOpenPage (pMyPort, NIL);
}

{ Here you image the current page by calling QuickDraw! The drawing proc will need the page size and printer resolution stored in the Print.PrInfo. }

PrMyPage (iPage, hPrint^^.PrInfo);

```
PrClosePage (pMyPort);
END;
PrCloseDoc (pMyPort):
```

Picture Printing

The Pic printing procs are the standard way to print. A third proc is provided to do simple bitmap printing.

```
PROCEDURE PrPicFile( hPrint: THPrint; pPrPort: TPPrPort; pIOBuf: Ptr; pOevBuf: Ptr; YAR PrStatus: TPrStatus );
```

This reads and prints the spooled print file. If an IdleProc is included in the Print record, it is run both during imaging and writing to the serial port. The first three parameters are identical to the PrOpenDoc parameters. The device buffer is the "band" buffer and associated data. If NIL, I allocate it. Its size is Print.PrXInfo.iDevBytes. The status record simply records the progress of printing and may be used by the IdleProc. See PrApp for its use.

```
PROCEDURE PrPic ( hPic: PicHandle; hPrint: THPrint; pPrPort: TPPrPort; pDevBuf: Ptr; VAR PrStatus: TPrStatus);
```

Simply prints from your picture rather than the spooled file.

```
PROCEDURE Proticall (iWhichCtl: Integer; iParami, iparami
```

iPrBitsCt1(= 4) —The bitmap printing control pBitMap: Ptr; —QuickDraw bitmap

pPortRect: TPRect; —a portrect. use bounds for mole bitm lControl: LongInt); —O=>Screen resolution. Portreit

..dumps a bitmap to the printer. IControl is a device dep param; use 0 for screen printing. Thus PrCtlCall (iPrBitsCtl, @MyPort^.ScreenBits, @MyPort^.PortRect.Bounds performs a screen dump of just my port's data.

Notes

Errors:

The alert reader will have noticed a lack of error returns from printing.

we can issue our own Alerts, we do so. Other errors are handled by posting the error in the printer globals. The first integer is the current error number. In case the user's disk does not have a printing resource file, or it is incorred named in System.rsrc, or there is no .Print driver; we simply post an error and No-op in the Prink code. We do not alert from Prink.

Size:

Printing is really a Mini-Application rather than a library. The code is tarrect roughly 8K. But this is really a small part of the cost of printing; even if the code were "free", the data used by printing can be huge. Current sizes [19] and the code were "free", the data used by printing can be huge.

```
Code: Print Driver = 780
PrLink = 374
Dialogs = 2,226
Spooling = 1,294
Draft = 2,134
Pic Printing = 4,630
```

Data: Bands = 6K for HiRes
Picture = 5K to 15K, typically. Max = 32K

Fonts = 3K to 10K, typically. Max = 32K
QuickDraw Buf = 2K to 6K, typically; up to 12K for 24 Pt shadow HiRes!
This is why a separate Printer Application is provided: you may simply spool and let the client use it. For spread sheets, for example, Draft printing may be adequate for most uses.

Bands:

The size of a HiRes page bitmap is in the order of 1/4 Megabyte! Printing handles this by breaking the page into smaller "bands" and alternates imaging and printing them to print a page. For example, a HiRes US Letter size page has 47 bands of 5120 bytes for a total of 240,640 bytes. LoRes is 24 bands of 2560 bytes for a total of 51,440. [Note: as resolution doubles, data volume quadruples!]

Spooling:

Even if you plan to try to print from within the App, spooling is useful! It allows you to have a very clean swapping stratagy: First you spool, with only 3K of printing code and no more than 1K data. This may require much of your code and data to be reside But when the data is spooled, you can swapp all of your code and data out and call PrPicFile from its own micro segment! The banding stratagy requires very fast imaging of the data if it is to be drawn 50 times per page. This is another reason for spooling: Pic drawing is optimal use of QuickDraw!

Oraft:

Draft printing is a compromise between Ascii/WriteLn/Fast printing and QuickDraw. One of the strongest Mac attitudes is the full use of the QuickDraw style of imaging. Note that there are NO WriteLn's available for Mac programmers, and NO programs use a command line interface! I decided that the best compromise was Draft printing. It is called "Draft" mainly because it "simulates" the output you will get when you print with standard printing. It is done by simply installing QuickDraw capture procs and translating them to the printer's command codes. It thus can provide full use of the printer's native capabilities, such as bold, underline, fonts, line plotting, etc. It also requires no special interface such as WriteLn; thus the standard Apps get it without even being aware its happening! .. Its completely under client control! "But what do I do if all I want to do is make a Listing?!" Well, its really not so bad: simply get a default Print record, and make your own WriteLn! The only real headache is having to be aware of the Page boundaries. But the advantage is that the results will nicely fit European paper sizes, and you'll probably find that "Pretty Printing" will be so easy that you'll provide it. Note that it also lets you provide Spooled standard printing by simply changing one flag!

Printo:

The Prinfo record contains the device dependent parameters for the current printer. If carefully used, it provides the Application with "parametric device independence"

TPrinto = RECORD

(Font mgr/QuickDraw device code)

iDev: Integer; iVRes: Integer:

(Resolution of device, in device coordinates)

iHRes: Integer;

..note: V before H => compatable with Fourt

rPage: Rect;

{The page (printable) rectangle in device coor access

END;

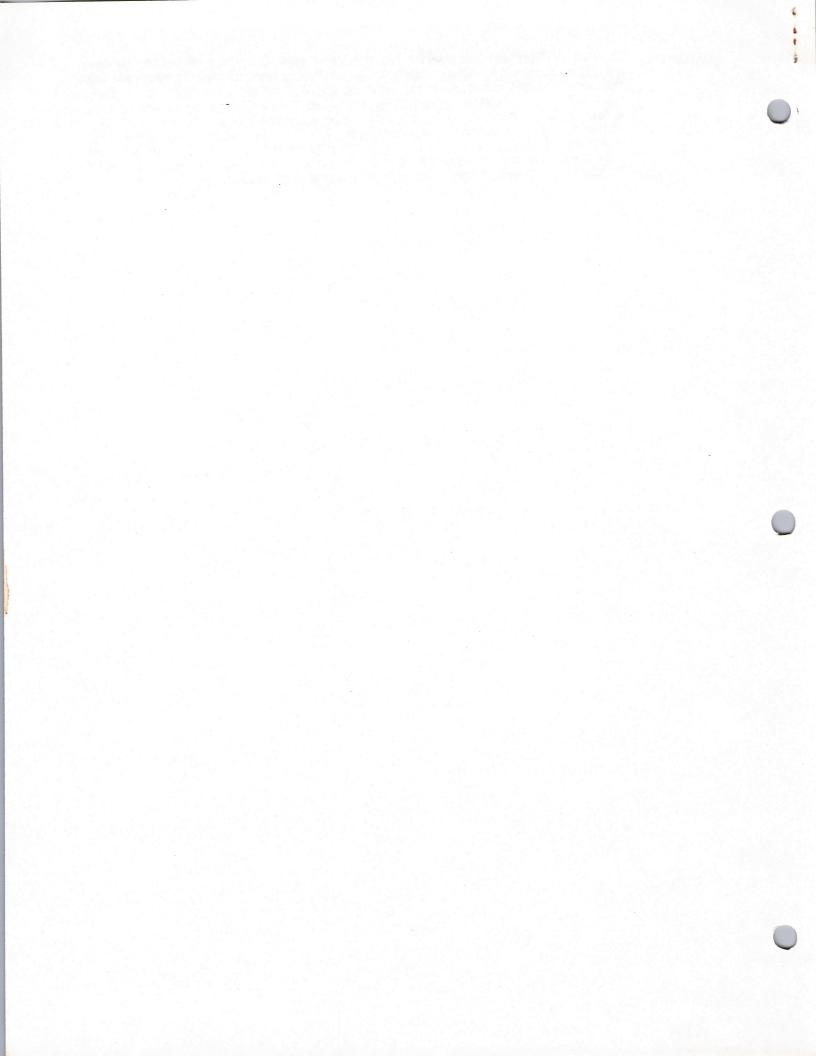
The most important field is the page rectangle. This gives the current sames will in bits. The next is the h/v resolution, in spots per inch. Finally, there is the QuickDraw - FontMgr device number. This lets you get the metrics for the will fonts, so that you can adjust for screen-printer differences. Correct use of these will result in a very surprising degree of printer independence.

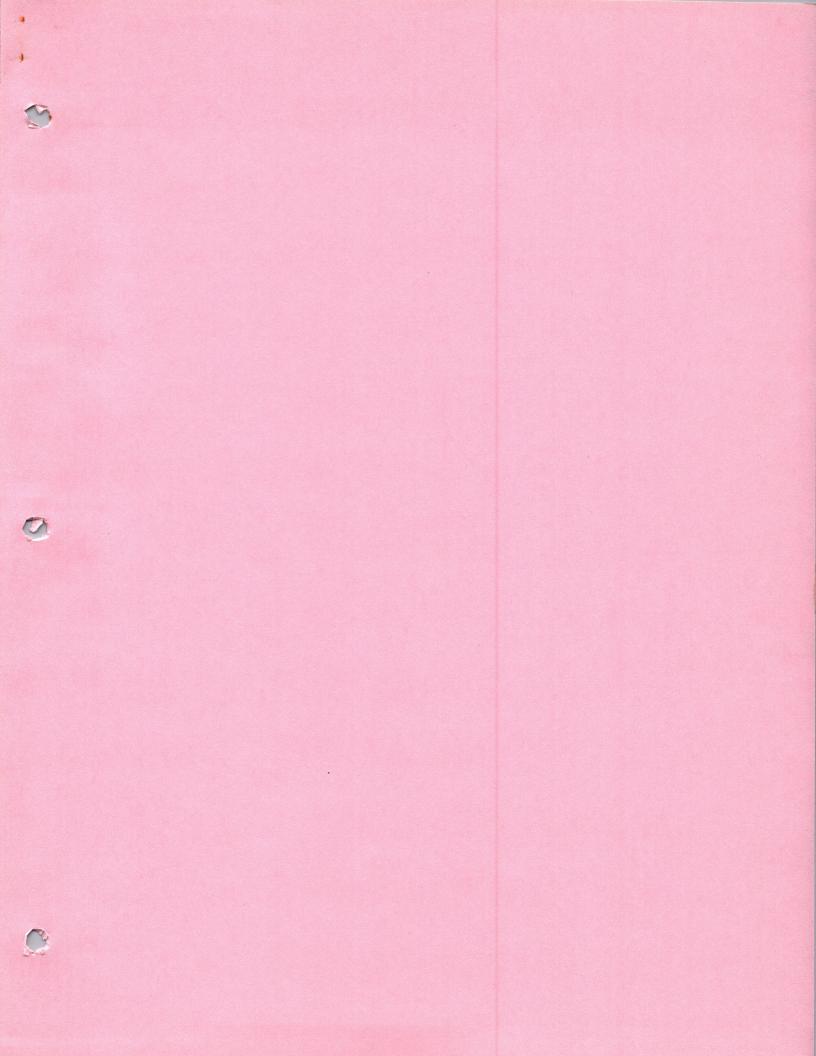
Idle:

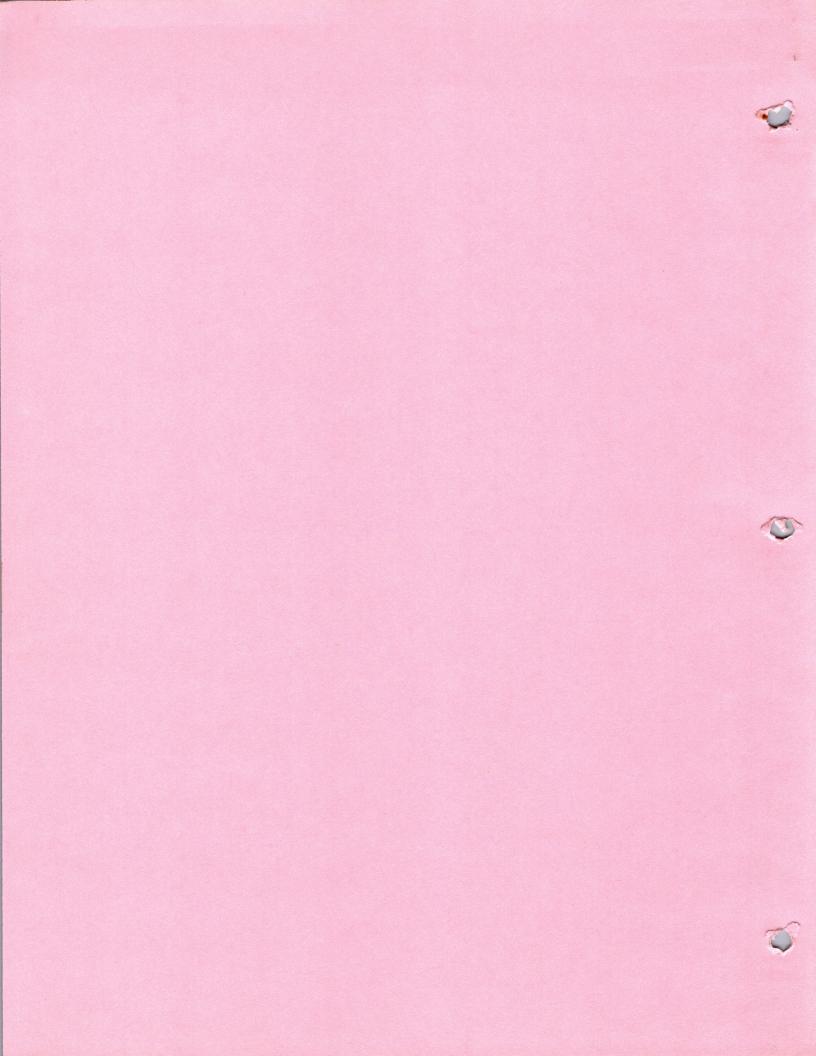
The Print record has a IdleProc: Print.PrJob.pldleProc. It is always returned to the default and dialog procs. To use it, simply stuff it with your own proc after the PrJobDialog and before calling the Draft or Pic Printing procs. A word of as this however! The "concurrancy" problems caused by the Idle proc are subtle and making Look at the PrTest and PrApp samples for how they do it. The major problem is making sure the GrafPort is reset to mine when returning from your idle. Also Drink Transport calling Printing procs while idling. They are accessed through the PrLink tipe is not re-entrant. The suggested idle proc is one polling for Cmd "." aborts. To apprepare the Print process of the Prin

Release:

Billions of files are released on the MacPrint disk, only four of which you need: PrLink to Link with, either MacPrint.obj or PrEqu.text to compile/assemble with, the current print resource to run with, and PrApp to let your client print spooled files with. New releases of printing simply use a new print resource, even if adding a new printer? Note however, that you must have the newest System.rsrc file which contains two vital printing resources: the .Print driver and a string containing the file name of the current printing resource [=ImageWriter]. If you have an older System.rsrc, it can be updated with these, using RMover, by pasting the small file PrSys.rsrc included in the printing release into System.rsrc.







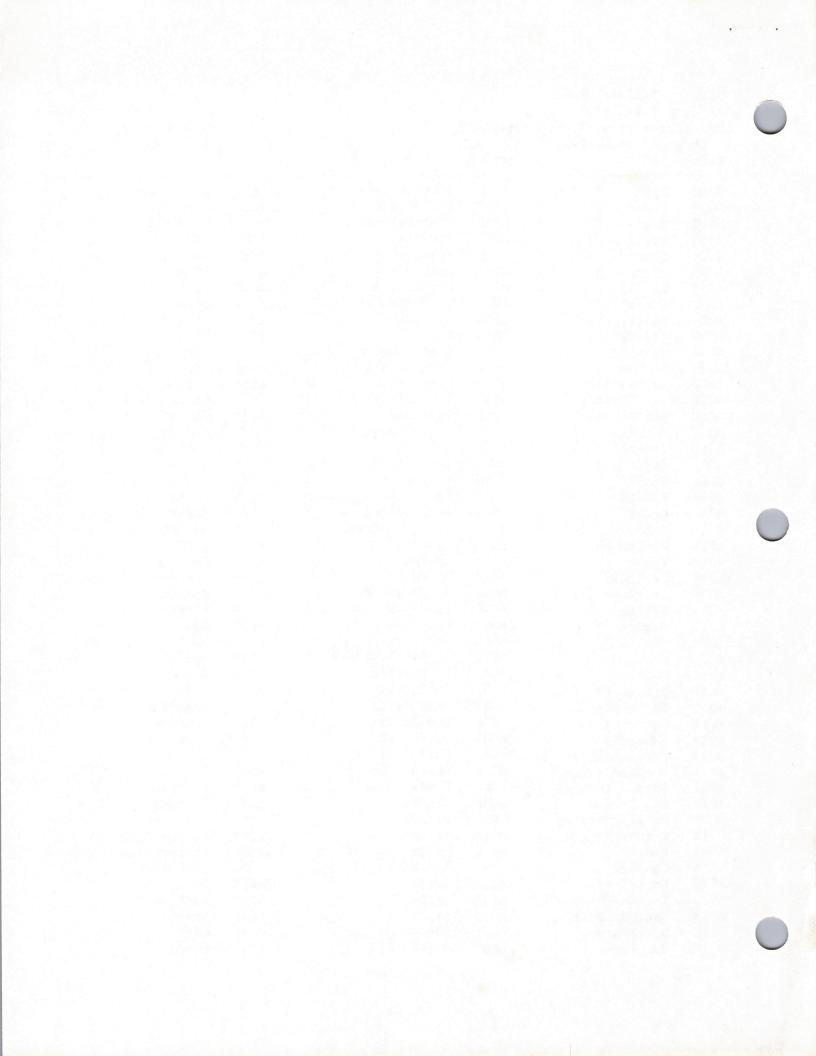
TOOLBOX NAMES

File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Name: Fields:

A000	Open	A030	OSEventAvail	AC63	BackColor	
A001	Close	A031	GetOSEvent	AC64	ColorBit	
A002	Read	A032	FlushEvents	AC65	GetPixel	
A003	Write	A033	VInstall	AC66	StuffHex	
A004	Control	A034	VRemove	AC67	LongMul	
A005	Status	A037	ReadParam	AC68	FixMul	
A006	K111I0	A038	WriteParam	AC69	FixRatio	
A007	GetVolInfo	A039	ReadDateTime	AC6A	HiWord	1
A008	FileCreate	A03A	SetDateTime	AC6B	LoWord	
A009	FileDelete	A03B	Delay	AC6C	FixRound	
AOOA	OpenRf	A03C	CmpString	AC6D	InitPort	
AOOB	Rename	A03D	DrvrInstall	AC6E	InitGraf	
A00C	GetFileInfo	A03E	DrvrRemove	AC6F		
AOOD	SetFileInfo	A03F	InitUtil	AC70		
A00E	UnmountVol	A040	ResrvMem	AC71		
A00F	MountVol	A041	SetFilLock	AC72	GrafDevice	
A010	FileAllocate	A042	RstFilLock	AC73	SetPort	
A011	GetEOF	A043	SetFilType	AC74	GetPort	
A012	SetEOF	A044	SetFPos	AC75	SetPortBits	
A013	FlushVol	A045	FlushFil	AC76	PortSize	
A014	GetVol	A046	GetTrapAddress	AC77		
A015	SetVol	A047	SetTrapAddress	AC78	SetOrigin	
A016	FInitQueue	A048	PtrZone	AC79	·	
A017	Eject	A049	HPurge	AC7A	GetClip	
A018	GetFPos	A04A	HNoPurge	AC7B	ClipRect	
A019	InitZone	A04B	SetGrowZone	AC7C	BackPat	
AOIA	GetZone	A04C	CompactMem	AC7D		
A01B	SetZone	AO4D	PurgeMem	AC7E		
A01C	FreeMem	A04E	AddDrive	AC7F		
AOID	MaxMem	A04F	InstallRDrivers	AC80		
AOIE	NewPtr	AC50	InitCursor	AC81	그리다 그리아 아이를 시간했다면, 그리아 시간에 얼굴하면서 그리고 그렇지 않는 그렇다.	
A01F	DisposePtr	AC51	SetCursor	AC82		
A020	SetPtrSize	AC52	HideCursor	AC83		
A021	GetPtrSize	AC53	ShowCursor	AC84	•	
A022	NWHandle	AC54	UprString	AC85	DrawText	
A023	DsposeHandle .	AC55	ShieldCursor	AC86	TextWidth	
A024	SetHandleSize	AC56	ObscureCursor	AC87	TextFont	
A025	GetHandleSize	AC57	SetApplBase	AC88		
A026	HandleZone	AC58	BitAnd	AC89		
A027	ReAllocHandle	AC59	BitXor	AC8A		
A028	RecoverHandle	AC5A	BitNot	AC8B		
A029	HLock	AC5B	BitOr	AC8C		
AO2A	HUnlock	AC5C	BitShift	AC8D		
A02B	EmptyHandle	AC5D	BitTst	AC8E		
A02C	InitApplZone	AC5E	BitSet	AC90		
AO2D	SetApplLimit	AC5F	BitClr	AC91	LineTo	
A02E	BlockMove	AC61	Random	AC92		
A02F	PostEvent	AC62	ForeColor	AC93	MoveTo	



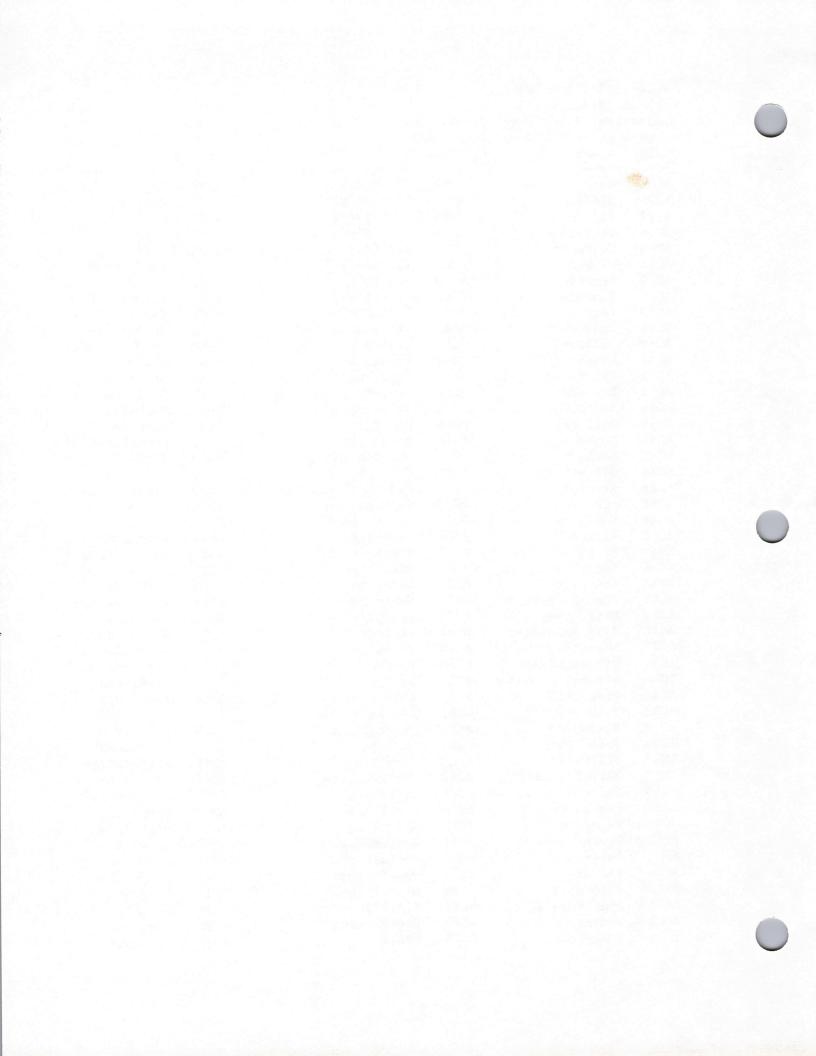
Page Feb 6, 198

File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Name: Fields:

-					
AC94	Moov	ACC7	PaintPoly	ACF9	MapPt
AC96	HidePen	ACC8	ErasePoly	ACFA	MapRect
AC97	ShowPen	ACC9	InvertPoly	ACFB	MapRgn
AC98	GetPenState	ACCA	FillPoly	ACFC	MapPoly
AC99	SetPenState ?	ACCB	OpenPoly	ACFE	InitFonts
AC9A	GetPen	ACCC	ClosePoly	ACFF	GetFontName
AC9B	PenSize	ACCD	KillPoly	ADO0	GetFNum
AC9C	PenMode	ACCE	OffsetPoly	AD01	FMSwapFont
AC9D	PenPat	ACCF	PackBits	ADO2	RealFont
AC9E	PenNormal	ACDO	UnPackBits	ADO3	SetFontLock
ACA0	StdRect	ACD1	StdRgn	ADO4	DrawGrowIcon
ACA1	FrameRect	ACD2	FrameRgn	AD05	DragGrayRgn
ACA2	PaintRect	ACD3	PaintRgn	AD06	NewString
ACA3	EraseRect	ACD4	EraseRgn	AD07	SetString
ACA4	InvertRect	ACD5	InvertRgn	AD08	ShowHide
ACA5	FillRect	ACD6	FillRgn .	AD09	CalcVis
ACA6	EqualRect	ACD8	NewRgn	ADOA	CalcVisBehind
ACA7	SetRect	ACD9	DisposeRgn	ADOB	ClipAbove
ACA8	OffsetRect	ACDA	OpenRgn	ADOC	PaintOne
ACA9	InsetRect	ACDB	CloseRgn	ADOD	PaintBehind
ACAA	SectRect	ACDC	CopyRgn	ADOE	SaveOld
ACAB	UnionRect	ACDD	SetEmptyRgn	ADOF	DrawNew
ACAC	Pt2Rect	ACDE	SetRectRgn	AD10	GetWMgrPort
ACAD	PtInRect	ACDF	RectRgn	AD11	CheckUpDate
ACAE	EmptyRect	ACE0	OffsetRgn	AD12	InitWindows
ACAF	StdRRect	ACE1	InsetRgn	AD13	NewWindow
ACB0	FrameRoundRect	ACE2	EmptyRgn	AD14	DisposeWindow
ACB1	PaintRoundRect	ACE3	EqualRgn	AD15	ShowWindow
ACB2	EraseRoundRect	ACE4	SectRgn	AD16	HideWindow
ACB3	InvertRoundRect	ACE5	UnionRgn	AD17	GetWRefCon
ACB4	FillRoundRect	ACE6	DiffRgn	AD18	SetWRefCon
ACB6	Std0val	ACE7	XOrRgn	AD19	GetWTitle
ACB7	FrameOval	ACE8	PtInRgn	ADIA	SetWTitle
ACB8	PaintOval	ACE9	RectInRg	AD1B	MoveWindow
ACB9	EraseOval	ACEA	SetStdProcs	AD1C	HiliteWindow
ACBA	InvertOval	ACEB	StdBits	AD1D	SizeWindow
ACBB	FillOval	ACEC	CopyBits	ADIE	TrackGoAway
ACBC	SlopeFromAngle	ACED	StdTxMeasure	ADIF	SelectWindow
ACBD	StdArc	ACEE	StdGetPic	AD20	BringToFront
ACBE	FrameArc .	ACEF	ScrollRect	AD21	Send Behind
ACBF	PaintArc	ACFO	StdPutPic	AD22	BeginUpdate
ACCO	EraseArc	ACF1	StdComment	AD23	
ACC1	InvertArc	ACF2	PicComment	AD24	EndUpdate
ACC2	FillArc	ACF3	OpenPicture		FrontWindow
ACC3	PtToAngle	ACF4	ClosePicture	AD25	DragWindow
ACC4	AngleFromSlope	ACF5	KillPicture	AD26	DragTheRgn
CC5	StdPoly	ACF6	DrawPicture	AD27	InvalRgn
LCC6	FramePoly	ACF8	ScalePt	AD28	InvalRect
-300	- rameroty	10010	Caler	AD29	ValidRgn



Page 3 Feb 6, 1984

File: ToolBox Names

Report: TrapList
Selection: Value/Trap: equals A000
through Value/Trap: equals AFFF

Value/	Name:	Fields:

AD2A	ValidRect	AD5C	SizeControl		AD8F	SetIText
AD2B	GrowWindow	AD5D	HiliteControl		AD90	GetIText
AD2C	FindWindow	AD5E	GetCTitle		AD91	ModalDialog
AD2D	CloseWindow	AD5F	SetCTitle		AD92	DetachResouce
AD2E	SetWindowPic	AD60	GetCtlValue		AD93	SetResPurge
AD2F	GetWindowPic	AD61	GetCtlMin		AD94	CurResFile
AD30	InitMenus	AD62	GetCt1Max		AD95	InitResources
AD31	NewMenu	AD63	SetCtlValue		AD96	RsrcZoneInit
AD32	DisposeMenu	AD64	SetCtlMin		AD97	OpenResFile
AD33	AppendMenu	AD65	SetCtlMax		AD98	UseResFile
AD34	ClearMenuBar	AD66	TestControl		AD99	UpdateResFile
AD35	InsertMenu	AD67	DragControl	- 1	AD9A	CloseResFile
AD36	DeleteMenu	AD68	TrackControl		AD9B	SetResLoad
AD37	DrawMenuBar	AD69	DrawControls		AD9C	CountResources
AD38	HiliteMenu	AD6A	GetCtlAction		AD9D	GetIndResource
AD39	EnableItem	AD6B	SetCtlAction		AD9E	CountTypes
AD3A	DisableItem	AD6C	FindControl	- 1	AD9F	GetIndType
AD3B	GetMenuBar	AD6E	DeQueue		ADA0	GetResource
AD3C	SetMenuBar	AD6F	EnQueue		ADA1	GetNamedResourc
AD3D	MenuSelect	AD70	GetNextEvent		ADA2	LoadResource
AD3E	MenuKey	AD71	EventAvail		ADA3	ReleaseResource
AD3F	GetItemIcon	AD72	GetMouse		ADA4	HomeResFile
AD40	SetItemIcon	AD73	StillDown		ADA5	SizeRsrc
AD41	GetItemStyle	AD74	Button		ADA6	GetResAttrs
AD42	SetItemStyle	AD75	TickCount		ADA7	SetResAttrs
AD43	GetItemMark	AD76	GetKeys		ADA8	GetResInfo
AD44	SetItemMark	AD77	WaitMouseUp		ADA9	SetResInfo
AD45	CheckItem	AD79	CouldDialog		ADAA	ChangedResData
AD46	GetItem	AD7A	FreeDialog		ADAB	AddResource
AD47	SetItem	AD7 B	InitDialogs		ADAC	AddReference
AD48	CalcMenuSize	AD7C	GetNewDialog		ADAD	RmveResource
AD49	GetMHandle	AD7D	NewDialog		ADAE	RmveReference
AD4A	SetMenuFlash	AD7E	SetIText		ADAF	ResError
AD4B	PlotIcon	AD7F	IsDialogEvent		ADB0	WriteResource
AD4C AD4D	FlashMenuBar	AD80	DialogSelect		ADB1	CreateResFile
AD4E	AddResMenu	AD81	DrawDialog		ADB2	SystemEvent
AD4F	PinRect	AD82	CloseDialog		ADB3	SystemClick
	DeltaPoint	AD83	DisposeDialog		ADB4	SystemTask
AD50 AD51	CountMItems	AD85	Alert		ADB5	SystemMenu
AD54	InsertResMenu	AD86	StopAlert		ADB6	OpenDeskAcc
AD55	NewControl	AD87	NoteAlert		ADB7	CloseDeskAcc
AD56	DisposeControl	AD88	CautionAlert		ADB8	GetPattern
AD57	KillControls	AD89	CouldAlert		ADB9	GetCursor
AD58	ShowControl	AD8A	FreeAlert		ADBA	GetString
AD59	HideControl	AD8B	ParamText		ADBB	GetIcon
AD5A	MoveControl GetCRefCon	AD8C	ErrorSound		ADBC	GetPicture
AD5B	SetCRefCon	AD8D	GetDItem		ADBD	GetNewWindow
טעעה	DE LUNET COLI	AD8E	SetDItem		ADBE	GetNewControl



File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF

Value/ Name:

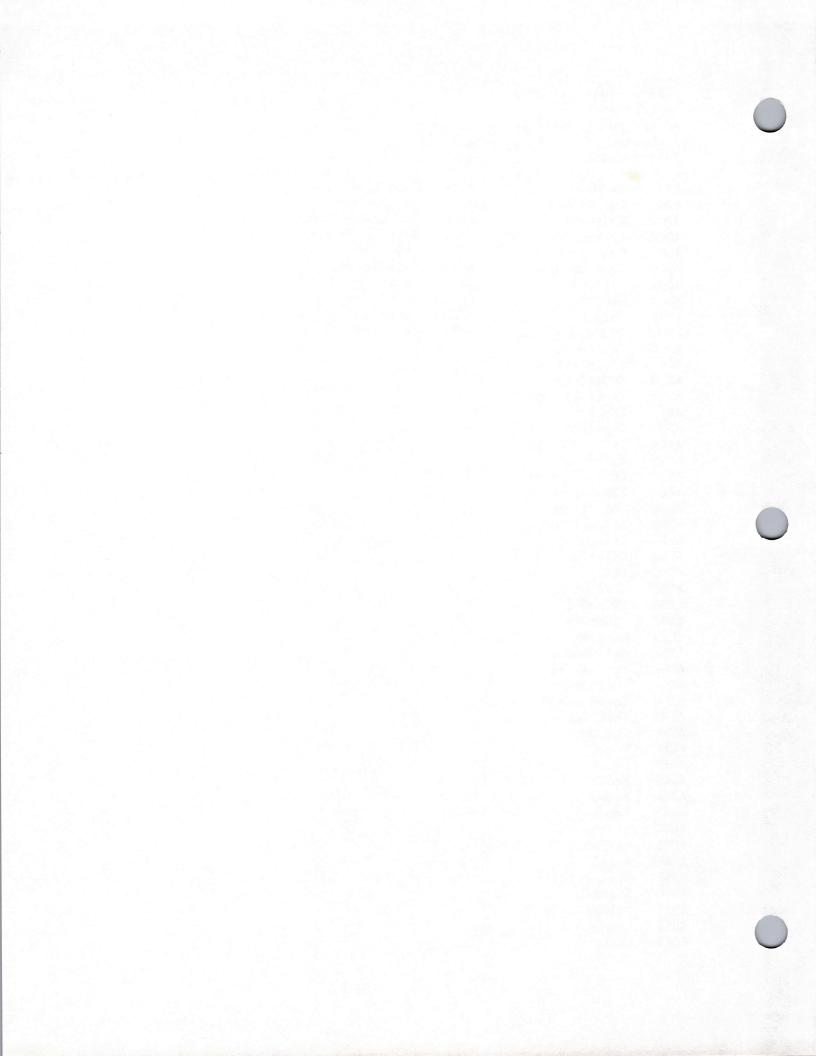
ADF3

Chain

Fields:

		nemet.	rielus.	
	ADBF	GetMenu	ADF4	ExitToShell
	ADC0	GetNewMBar	ADF5	GetAppParms
	ADC1	UniqueID	ADF6	GetResFileAttrs
	ADC2	SystemEdit	ADF7	SetResFileAttrs
	ADC8	SystemBeep	ADF9	InfoScrap
	ADC9	SystemError	ADFA	UnloadScrap
	ADCA	PutIcon	ADFB	LoadScrap
	ADCB	TeGetText	ADFC	ZeroScrap
	ADCC	TEInit	ADFD	GetScrap
	ADCD	TEDispose	ADFE	PutScrap
	ADCE	TextBox		
	ADCF	TESetText		
	ADD0	TECalText		
	ADD1	TESetSelect		•
	ADD2	TENew		
	ADD3	TEUpdate		
	ADD4	TEClick		
	ADD5	TECopy		
	ADD6	TECut		
	ADD7	TEDelete		
	ADD8	TEActivate		
	ADD9	TEDeactivate		
	ADDA	TEIdle		
	ADDB	TEPaste		
	ADDC	TEKey		
	ADDD	TEScrol1		
	ADDE	TEInsert		
	ADDF	TESetJust		
	ADE0	Munger		
	ADE1	HandToHand		
	ADE2	PtrToXHand		
	ADE3	PtrToHand		
	ADE4	HandAndHand		
	ADE5	InitPack		
	ADE6	InitMath		
	ADE8	Pack0 Pack1		
	ADE9			
	ADEA	Pack2		
	ADEB	Pack3 Pack4		
	ADEC	Pack5		
	ADED			
	ADEE	Pack6		
	ADEF	Pack7		
		PtrAndHand		
		LoadSeg		
		UnLoad Seg		
-	ADE2	Launch		

Page 4 Feb 6, 1984

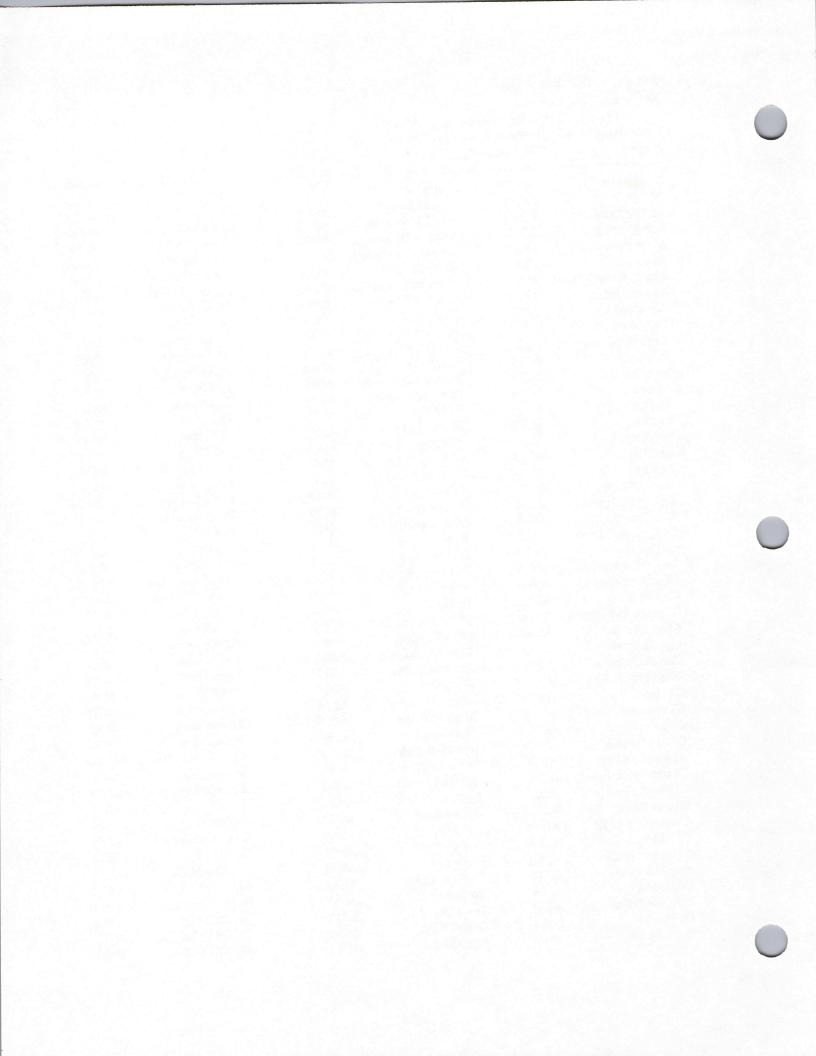


Page 1 Feb 6, 1984

File: ToolBox Names Renort: TrapList

s ction: Value/Trap: equals A000 chrough Value/Trap: equals FFFF Name: Value/ Fields:

	AddDrive	A04E	Canal				
	AddPt	AC7E	CopyRgn CouldAlert	ACDC	EraseArc	ACCO	
	AddReference	ADAC	Confidater	AD89	EraseOval	ACB9	
	AddResMenu	AD4D	CouldDialog	AD79 1	ErasePoly	ACC8	
	AddResource	ADAB	CountMItems	AD50	EraseRect	ACA3	
	Alert	AD85	CountResource		EraseRgn	ACD/	
	AngleFromSlop	e ACC4	CountTypes	AD9E	EraseRoundRed	ct ACB2	
	AppendMenu	AD33	CreateResFile		ErrorSound	AD8C	
	BackColor	AC63	CurResFile Delay	AD94	EventAvail	AD71	
	BackPat	AC7C		A03B	ExitToShell	ADF4	
	BeginUpdate	AD22	DeleteMenu DeltaPoint	AD36	FileAllocate	A010	
	BitAnd	AC58	DeQueue	AD4F	FileCreate	A008	
	BitClr	AC5F		AD6E	FileDelete	A009	
	BitNot	AC5A	DetachResouce		FillArc	ACC2	
*	BitOr	AC5B	DialogSelect	AD80.	FillOval	ACBB	
	BitSet	AC5E	DiffRgn	ACE6	Fill Poly	ACCA	
	BitShift	AC5C	DisableItem	AD3A	FillRect	ACA5	
	BitTst	AC5D	DisposeControl	경이님, 그림 왕이 됐다.구나 그 방송의 편네.	FillRgn	ACD6	
	BitXor		DisposeDialog	AD83	FillRoundRect	ACB4	
	BlockMove	AC59 A02E	DisposeMenu	AD32	FindControl		
	BringToFront	AD20	DisposePtr	A01F	FindWindow	AD6C	
	P on	AD74	DisposeRgn	ACD9	FInit Queue	AD2C	
	C MenuSize	AD48	DisposeWindow	AD14	FixMul	A016	
	CalcVis		DragControl	AD67	FixRatio	AC68	
	CalcVisBehind	ADO9	DragGrayRgn	AD05	FixRound	AC69 AC6C	
	CautionAlert	ADOA	DragTheRgn	AD26	FlashMenuBar		
	Chain	AD88	DragWindow	AD25	FlushEvents	AD4C	
	ChangedResData	ADF3	DrawChar	AC83	FlushFi1	A032	
	CharWidth	ADAA	DrawControls	AD69	FlushVol	A045	
	CheckItem	AC8D AD45	DrawDialog	AD81	FMSwapFont	A013 AD01	
4.19			DrawGrowIcon	ADO4	ForeColor		
	ClearMenuBar	AD11	DrawMenuBar	AD37	FrameArc	AC62	
	ClipAbove	AD34	DrawNew	ADOF	FrameOval	ACBE	
	ClipRect	ADOB	DrawPicture	ACF6	FramePoly	ACB7	
	Close	AC7B	DrawString	AC84	FrameRect	ACC6	
	CloseDeskAcc	A001	DrawText	AC85	FrameRgn	ACA1	
	CloseDeskacc	ADB7	DrvrInstall	AO3D	FrameRoundRect	ACD2	
	CloseDialog ClosePicture	AD82	DrvrRemove	A03E	FreeAlert	ACB0	
	ClosePoly	ACF4	DsposeHandle	A023	FreeDialog	AD8A	
	01 5	ACCC	Eject	A017	FreeMem	AD7A	
	C1 2	AC7D	EmptyHandle	AO2B		A01C	
		AD9A	EmptyRect	ACAE	FrontWindow	AD24	
	C1	ACDB	EmptyRgn	ACE2	GetAppParms	ADF5	
	C C	AD2D	Frall T.	AD39	GetClip	AC7A	
	C-1- N.	A03C	P- 111 1 .	AD23	GetCRefCon	AD5A	
		AC64	F-0	AD6F	GetCTitle	AD5E	
		A04C	P	AC81	GetCtlAction	AD6A	
		A004	Faural D.	ACA6	GetCtlMax	AD62	
-	its	ACEC	Fau - 1 D	ACE3	GetCtlMin	AD61	
				nce)	GetCtlValue	AD60	

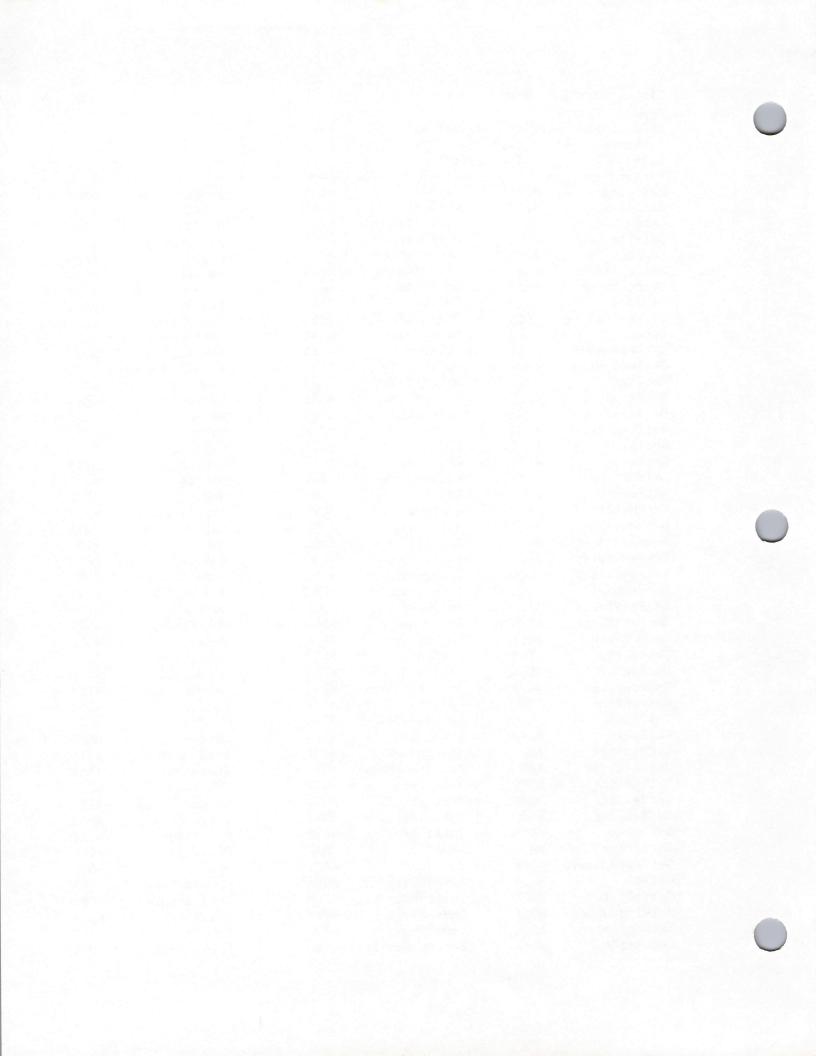


File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals FFFF Value/ Fields: Name:

-	-				~~~~~
GetCursor	ADB9	GetWTitle	AD19	IsDi alogEvent	AD7F
GetDItem	AD8D	GetZone	A01A	KillControls	AD56
GetEOF	A011	GlobalToLocal	AC71	K111I0	A006
GetFileInfo	A00C	Graf Device	AC72	KillPicture	ACF5
GetFNum	AD00	GrowWindow	AD2B	KillPoly	ACCD
GetFontInfo	AC8B	Hand And Hand	ADE4	Launch	ADF2
GetFontName	ACFF	HandleZone	A026	Line	AC92
GetFPos	A018	HandToHand	ADE1	LineTo	AC91
GetHandleSize	A025	HideControl	AD58	LoadResource	ADA2
GetIcon	ADBB	HideCursor	AC52	LoadScrap	ADFB
Get Ind Resource	AD9D	HidePen	AC96	LoadSeg	ADF0
Get Ind Type	AD9F	HideWindow	AD16	LocalToGlobal	AC70
GetItem	AD46	HiliteControl	AD5D	LongMul	AC67
GetItemIcon	AD3F	HiliteMenu	AD38	LoWord	AC6B
GetItemMark	AD43	HiliteWindow	AD1C	MapPoly	ACFC
GetItemStyle	AD41	HiWord	AC6A	MapPt	ACF9
GetIText	AD90	HLock	A029	MapRect	ACFA
GetKeys	AD76	HNoPurge	A04A	MapRgn	ACFB
GetMenu	ADBF	HomeResFile	ADA4	MaxMem	AO1D
GetMenuBar	AD3B	HPurge	A049	MenuKey	AD3E
GetMHandle	AD49	HUnlock	A02A	MenuSelect	AD3D
GetMouse	AD72	InfoScrap	ADF9	ModalDialog	AD91
GetNamedResourc		InitApplZone	A02C	Moov	AC94
GetNewControl	ADBE	InitCursor	AC50	MountVol	AOOF
GetNewDialog	AD7C	InitDialogs	AD7B	MoveControl	AD59
GetNewMBar	ADC0	InitFonts	ACFE	MovePortTo	AC77
GetNewWindow	ADBD	InitGraf	AC6E	MoveTo	AC93
GetNextEvent	AD70	InitMath	ADE6	MoveWindow	
GetOSEvent	A031	InitMenus	AD30	Munger	ADIB
GetPattern	ADB8	InitPack	ADE5	NewControl	ADE0
GetPen	AC9A	InitPort	AC6D	NewDialog	AD54
GetPenState	AC98	InitResources	AD95	NewMenu	AD7D
GetPicture	ADBC	InitUtil	AO3F		AD31
GetPixel	AC65	InitWindows	AD12	NewPtr	A01E
GetPort	AC74	InitZone	A019	NewRgn	ACD8
GetPtrSize	A021	InsertMenu	AD35	NewString	AD06
GetResAttrs	ADA6	InsertResMenu	AD51	NewWindow	AD13
GetResFileAttrs		InsetRect	ACA9	NoteAlert	AD87
GetResInfo	ADA8			NWHandle	A022
GetResource	ADA0	InsetRgn	ACE1	ObscureCursor	AC56
GetScrap	ADFD	InstallRDrivers		OffsetPoly	ACCE
GetString		InvalRect	AD28	OffsetRect	ACA8
GetTrapAddress	ADBA	InvalRgn	AD27	OffsetRgn	ACE0
GetVol	A046	InvertArc	ACC1	0pen	A000
GetVolInfo	A014	InvertOval	ACBA	OpenDeskAcc	ADB6
GetWindowPic	A007	InvertPoly	ACC9	OpenPicture	ACF3
	AD2F	InvertRect	ACA4	OpenPoly	ACCB
GetWMgrPort	AD10	InvertRgn	ACD5	OpenPort	AC6F
GetWRefCon	AD17	InvertRoundRect	ACB3	OpenResFile	AD97

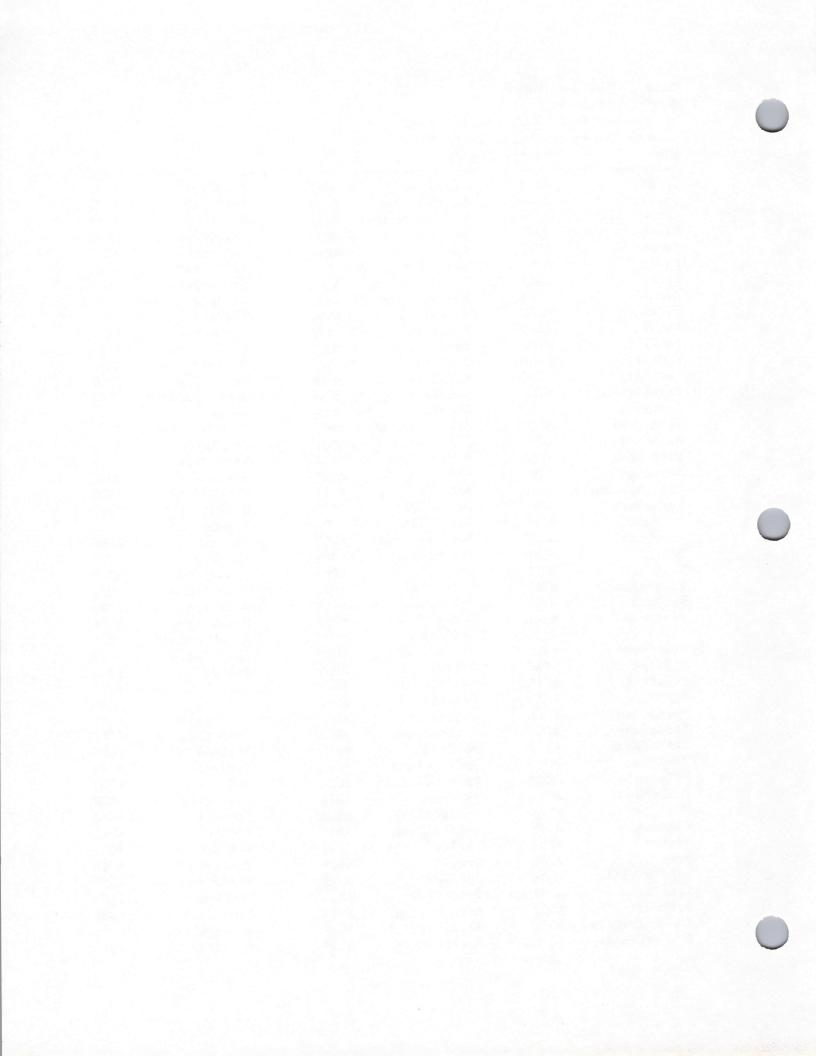


ToolBox Names ' Report: TrapList

ection: Value/Trap: equals A000 chrough Value/Trap: equals FFFF wate: Value/ Fields:

Name:

differential constant was not constant to the constant of					
OpenRf	AOOA	RectInRg	ACE9	SetPort	AC73
OpenRgn	ACDA	RectRgn	ACDF	SetPortBits	AC75 .
OSEventAvail	A 030	ReleaseResource	ADA3	SetPt	AC80
Pack0	ADE7	Rename	AOOB	SetPtrSize	A020
Packl	ADE8	ResError	ADAF	SetRect	ACA7
Pack2	ADE9	ResrvMem	A040	SetRectRgn	ACDE
Pack3	ADEA	RmveReference	ADAE	SetResAttrs	ADA7
Pack4	ADEB	RmveResource	ADAD	SetResFileAttrs	
Pack5	ADEC	RsrcZoneInit	AD96	SetResInfo	ADA9
Pack6	ADED	RstFillock	A042	SetResLoad	AD9B
Pack7	ADEE	SaveOld	ADOE	SetResPurge	AD93
PackBits	ACCF	ScalePt	ACF8	SetStdProcs	ACEA
PaintArc	ACBF	ScrollRect	ACEF	SetString	AD07
PaintBehind	ADOD	SectRect	ACAA ·	SetTrapAddress	A047
PaintOne	ADOC	SectRgn	ACE4	SetVol	A015
PaintOval	ACB8	SelectWindow	AD1 F	SetWindowPic	AD2E
PaintPoly	ACC7	Send Behind	AD21	SetWRefCon	AD18
PaintRect	ACA2	SetApplBase	AC57	SetWTitle	ADIA
PaintRgn	ACD3	SetApplLimit	A02D	SetZone	AO1B
PaintRoundRect	ACB1	SetClip	AC79	ShieldCursor	AC55
ParamText	AD8B	SetCRefCon	AD5B	ShowControl	AD57
P Mode	AC9C	SetCTitle	AD5F	ShowCursor	AC53
formal	AC9E	SetCtlAction	AD6B	ShowHide	ADO8
PenPat	AC9D	SetCtlMax	AD65	ShowPen	
PenSize	AC9B	SetCtlMin	AD64		AC97
PicComment	ACF2	SetCtlValue	AD63	ShowWindow	AD15
PinRect	AD4E	SetCursor	AC51	SizeControl	AD5C
PlotIcon	AD4B	SetDateTime	A03A	SizeRsrc	ADA5
PortSize	AC76	SetDItem	AD8E	SizeWindow	AD1D
PostEvent	A02F	SetEmptyRgn	ACDD	SlopeFromAngle	ACBC
Pt2Rect	ACAC	SetEOF	A012	SpaceExtra	AC8E
PtInRect	ACAD	SetFileInfo		Status	A005
PtInRgn	ACE8	SetFilLock	AOOD .	StdArc	ACBD
PtrAndHand	ADEF		A041	StdBits	ACEB
PtrToHand	ADE3		A043	StdComment	ACF1
PtrToXHand	ADE2	SetFontLock	AD03	StdGetPic	ACEE
PtrZone	A048	어느, 그렇게 프로그램 내용 가는 사람들이 되었다면 하는 것이 되었다.	A044	StdLine	AC90
PtToAngle	ACC3	a '	A04B	Std0val	ACB6
PurgeMem		그 이 그를 되고 하는데 하고요? 회사에 와서 그림은 그는 이상을 다른데 하면 없다.	A024	StdPoly	ACC5
utlcon	AO4D		AD47	StdPutPic	ACF0
utScrap	ADCA		AD40	StdRect	ACA0
Candom	ADFE		AD44	StdRgn	ACD1
	AC61		AD42	StdRRect	ACAF
lead	A002		AD7E	StdText	AC82
eadDateTime	A039		AD8F	StdTxMeasure	ACED
eadParam	A037		AD3C	StillDown	AD73
ealFont	AD02		AD4A	StopAlert	AD86
eAllocHandle	A027		AC78	StringWidth	AC8C
rerHandle	A028			StuffHex	AC66



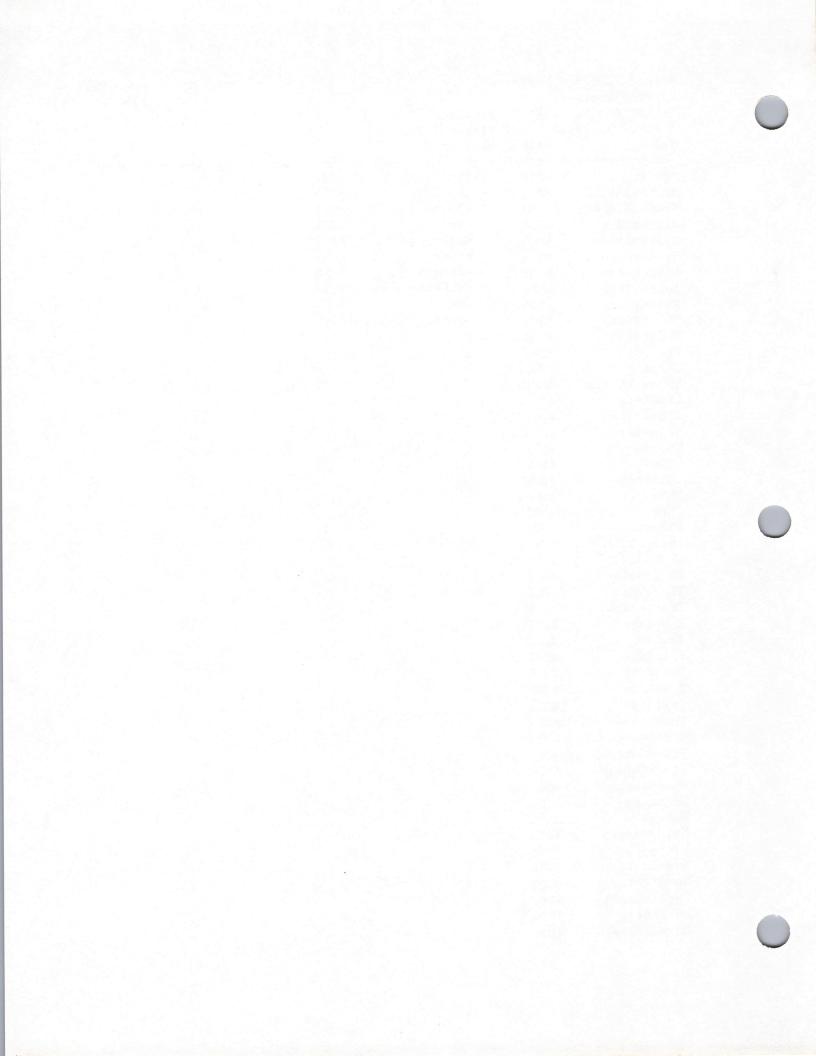
File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals FFFF was: Value/ Fields:

Name:

Mamc .	value/	rielas:	
SubPt	AC7F	ValidRect	AD2A
SystemBeep	ADC8	ValidRgn	AD29
SystemClick	ADB3	VInstall	A033
SystemEdit	ADC2	VRemove	A034
SystemError	ADC9	WaitMouseUp	AD77
SystemEvent	ADB2	Write	A003
SystemMenu	ADB5	WriteParam	A038
SystemTask	ADB4	WriteResource	ADBO
TEActivate	ADD8	XOrRgn	ACE7
TECalText	ADDO	ZeroScrap	ADFC
TEClick	ADD4		
TECopy	ADD5		
TECut	ADD6		
TEDeactivate	ADD9		
TEDelete	ADD7		
TEDispose	ADCD		
TeGetText	ADCB		
TEIdle	ADDA		
TEInit	ADCC		
TEInsert	ADDE		
TEKey	ADDC		
TENew	ADD2		
TEPaste	ADDB		
TEScrol1	ADDD		
TESetJust	ADDF		
TESetSelect	ADD1		
TESetText	ADCF		
TestControl	AD66		
TEUpdate	ADD3		
TextBox	ADCE		
TextFace	AC88		
TextFont	AC87		
TextMode	AC89		
TextSize	AC8A		
TextWidth	AC86		
TickCount	AD75		
TrackControl	AD68		
TrackGoAway	AD1E		
UnionRect	ACAB		
UnionRgn	ACE5		
UniqueID	ADC1		
UnloadScrap	ADFA		
UnLoad Seg	ADF1		
UnmountVol	AOOE		
UnPackBits	ACDO		
UpdateResFile	AD99		
UprString	AC54		
UseResFile	AD98		



File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000

through Value/Trap: equals AFFF
Value/ Name: Fields:

```
A000
       Open
       Close
A001
A002
       Read
A003
       Write
A004
       Control
A005
       Status
A006
       KillIO
A007
       GetVolInfo
A008
       FileCreate
A009
       FileDelete
A00A
       OpenRf
A00B
       Rename
A00C
       GetFileInfo
AOOD
       SetFileInfo
A00E
       Unmount Vol
A00F
       MountVol
A010
       FileAllocate
A011
       GetEOF
A012
       SetEOF
A013
       FlushVol
A014
       Get Vol
A015
       Set Vol
A016
       FInitQueue
A017
       Eject
A018
       GetFPos
A019
       InitZone
                       startPtr(id4) limitPtr(id4) masterCount(id2) growZone(ipc4)
A01A
       GetZone
                       theZone(odA0)
A01B
       SetZone
                       theZone(idA0)
A01C
       FreeMem
                       BytesFree(odD0)
AO1D
       MaxMem
                       MaxBytes(odD0)
A01E
       NewPtr
                       byteCount(idD0) Pointer(odA0)
A01F
       DisposePtr
                       Pointer(idA0)
A020
       SetPtrSize
                       Pointer(idA0) Size(idD0)
A021
       GetPtrSize
                       Pointer(idA0) Size(odD0)
A022
       NewHandle
                       BytesNeeded(idD0) Handle(odA0)
A023
       DsposeHandle
                       Handle(A0)
A024 SetHandleSize
                       Handle(AO) Size(DO)
A025
       GetHandleSize
                       Handle(A0) Size(D0)
A026 HandleZone
                       Handle(A0) HeapPtr(A0)
A027
       ReAllocHandle
                       Handle(A0) Size(D0)
A028
       RecoverHandle
                       Pointer(A0) Handle(A0)
A029 HLock
                       Handle(A0)
A02A
       HUnlock
                       Handle(A0)
A02B EmptyHandle
                       Handle(ihAO)
A02C
       InitApplZone
A02D
       SetApplLimit
A02E
       BlockMove
                       SourcePtr(A0) DestPtr(A1) Count(D0)
A02F
       PostEvent
```

File: ToolBox Names Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF

Value/ Name: Fields: OSEventAvail A030 GetOSEvent A031 A032 FlushEvents VInstall A033 VRemove A034 A037 ReadParam A038 WriteParam ReadDateTime A039 SetDateTime A03A A03B Delay CmpString A03C AO3D DrvrInstall DrvrRemove A03E A03F InitUtil A041 SetFilLock A042 RstFilLock SetFilType A043 A044 SetFPos A045 FlushFil A046 GetTrapAddress A047 SetTrapAddress theZone(ihA0) A048 PtrZone A049 Handle(A0) HPurge Handle(A0) A04A HNoPurge SetGrowZone AO4B AO4C CompactMem AO4D PurgeMem AddDrive AO4E AO4F InstallDrivers InitCursor AC50 SetCursor cursor(4) AC51 AC52 HideCursor AC53 ShowCursor AC54 UprString ShieldCursor AC55 AC56 ObscureCursor SetApplBase AC57 long1(4) long2(4) AC58 BitAnd long1(4) long2(4) AC59 BitXor long(4) AC5A BitNot long1(4) long2(4) AC5B BitOr AC5C BitShift bytePtr(4) bitNum(4) AC5D BitTst bytePtr(4) binNum(4) AC5E BitSet bytePtr(4) bitNum(4) AC5F BitClr Random AC61 color(4) AC62 ForeColor color(4) AC63 BackColor

File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Name: Fields:

ColorBit AC64 whichBit(2) GetPixel AC65 h(2) v(2)AC66 StuffHex thingPtr(4) s(2)AC67 LongMul a(4) b(4) dst(4)AC68 FixMul a(4) b(4)numerator(2) denominator(2) AC69 FixRatio AC6A HiWord x(4)AC6B LoWord x(4)x(4)AC6C FixRound AC6D InitPort grafpointer(4) AC6E InitGraf globalPtr(4) AC6F OpenPort grafPointer(4) AC70 LocalToGlobal pt(4) AC71 GlobalToLocal pt(4) AC72 GrafDevice device(2) AC73 SetPort grafPtr(4) AC74 GetPort grafPtr(4) AC75 SetPortBits theBitMap(4) AC76 PortSize width(2) height(2) AC77 MovePort leftGlobal(2) topGlobal(2) AC78 SetOrigin h(2) v(2)rgn(4) AC79 SetClip AC7A GetClip rgn(4) AC7B ClipRect r(4) AC7C BackPat pat(4) AC7D ClosePort grafPointer(4) AC7E AddPt source(4) dest(4) AC7F SubPt source(4) dest(4) AC80 SetPt pt(4) h(2) v(2)AC81 ptA(4) ptB(4) EqualPt AC82 StdText count(2) textAddr(4) AC83 DrawChar ch(2)AC84 DrawString s(4)AC85 DrawText textBuf(4) firstBute(2) byteCount(2) AC86 TextWidth textBuf(4) firstByte(2) byteCount(2) AC87 TextFont font(2) AC88 TextFace face(2) AC89 TextMode mode(2) AC8A TextSize size(2)AC8B GetFontInfo info(4)AC8C StringWidth s(4)AC8D CharWidth ch(2) AC8E SpaceExtra extra(2) newPt(4) AC90 StdLine AC91 LineTo h(2) v(2)AC92 Line dh(2) dv(2)AC93 MoveTo h(2) v(2)AC94 Moove dh(2) dv(2)

File: ToolBox Names Page 4
Report: TrapList July 14 1983

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Name: Fields:

```
Value/ Name:
                      Fields:
经存货收益 医存货物的存储的复数电影电影电影
AC96
      HidePen
AC97
      ShowPen
AC98
      GetPenState
                       pnState(4)
AC99
      SetPenState
                       pnState(4)
AC9A
      GetPen
                       pt(4)
AC9B
      PenSize
                       width(2) height(2)
AC9C
      PenMode
                       mode(2)
AC9D
     PenPat
                       pat(4)
AC9E
      PenNormal
ACA0
      StdRect
                       verb : GrafVerb ; r : Rect
      FrameRect
                       r: Rect
ACA1
ACA2
      PaintRect
                       r: Rect
ACA3
      EraseRect
                       r: Rect
ACA4
                       r: Rect
       InvertRect
                       r: Rect; pat: Pattern
ACA5
      FillRect
     EqualRect
ACA6
                       VAR r: Rect; left, top, right, bottom: Integer
ACA7
      SetRect
ACA8
       OffsetRect
                       VAR r: Rect; dh, dv: Integer
ACA9
       InsetRect
                       VAR r: Rect; dh, dv: Integer
ACAA
       SectRect
                       srcl, src2: Rect; VAR dst: Rect
                       srcl, src2: Rect; VAR dst: Rect
ACAB
       UnionRect
                       ptl, pt2: Point: VAR dst: Rect
ACAC
      Pt2Rect
ACAD
      PtInRect
                       pt: Point; r: Rect
ACAE
      EmptyRect
                       verb : GrafVerb ; r : Rect; ovW, ovH : INTEGER
ACAF
      StdRRect
      FrameRoundRect r: Rect; ovalWidth, ovalHeight: Integer
ACB0
      PaintRoundRect r: Rect; ovalWidth, ovalheight: Integer
ACB1
ACB2
      EraseRoundRect r: Rect: ovalWidth, ovalheight: Integer
ACB3
       InvertRoundRect r: Rect; ovalWidth, ovalHeight: Integer
ACB4
      FillRoundRect r: Rect; ovalWidth, ovalHeight: Integer; pat: Pattern
ACB6
      StdOval
                       verb : GrafVerb ; r : Rect
ACB 7
      FrameOval
                       r: Rect
ACB8
      PaintOval
                       r: Rect
                       r: Rect
ACB9
      EraseOval
       InvertOval
                       r: Rect
ACBA
                       r: Rect; pat: Pattern
ACBB
      FillOval
                       verb : GrafVerb ; r : Rect ; startAngle, arcAngle : INTEGER
ACBD
       StdArc
                       r : Rect; startAngle, arcAngle : INTEGER
ACBE FrameArc
                       r : Rect; startAngle, arcAngle : INTEGER
ACBF
      PaintArc
ACC0
      EraseArc
                       r : Rect; startAngle, arcAngle : INTEGER
                       r : Rect; startAngle, arcAngle : INTEGER
ACC1
       InvertArc
                       r : Rect; startAngle, arcAngle : INTEGER; pat : pattern
ACC2
       FillArc
                       r : Rect; pt : Point; VAR angle : INTEGER
ACC3
       PtToAngle
                       verb : GrafVerb ; poly : PolyHandle
ACC5
       StdPoly
ACC6
       FramePoly
                       poly: PolyHandle
ACC7
                       poly: PolyHandle
       PaintPoly
ACC8
                       poly: PolyHandle
       ErasePoly
                      poly: PolyHandle
ACC9
       InvertPoly
```

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Name: Fields: poly: PolyHandle; pat: Pattern FillPoly ACCA OpenPoly ACCB ClosePoly ACCC ACCD KillPoly poly: PolyHandle poly: PolyHandle; dh, dv: Integer ACCE OffsetPoly ACCF PackBits ACD0 UnPackBits ACD1 StdRgn verb : GrafVerb ; rgn : RgnHandle ACD2 FrameRgn rgn: RgnHandle rgn: RgnHandle ACD3 PaintRgn ACD4 EraseRgn rgn: RgnHandle ACD5 InvertRgn rgn: RgnHandle ACD6 FillRgn rgn: RgnHandle; pat: Pattern ACD8 NewRgn ACD9 DisposeRgn rgn: RgnHandle ACDA OpenRgn ACDB CloseRgn rgn: RgnHandle ACDC CopyRgn srcRgn, dstRgn: RgnHandle ACDD SetEmptyRgn ACDE SetRectRgn rgn: RgnHandle; left, top, right, bottom: Integer; ACDF rgn : RgnHandle; r : Rect RectRgn ACE0 OffsetRgn rgn: RgnHandle; dh, dv: Integer ACE1 InsetRgn rgn: RgnHandle; dh, dv: Integer ACE2 EmptyRgn rgn: RgnHandle ACE3 EqualRgn rgnA rgnB: RgnHandle ACE4 SectRgn srcRgnA, srcRgnB, dstRgn: RgnHandle ACE 5 UnionRgn srcRgnA, srcRgnB, dstRgn: RgnHandle ACE6 DiffRgn srcRgnA, srcRgnB, dstRgn: RgnHandle ACE7 XOrRgn srcRgnA, srcRgnB, dstRgn: RgnHandle ACE8 Pt InRgn pt: Point; rgn: RgnHandle ACE9 r:Rect; rgn: RgnHandle RectInRgn ACEA SetStdProcs ACEB StdBits VAR scrBits : BitMap; VAR srcRect, dstRect : Rect; mode : Inte ACEC CopyBits ACED StdTxMeasure ACEE StdGetPic ACEF ScrollRect dstRect: Rect; dh, dv: Integer; updateRgn: rgnHandle ACF0 StdPutPic ACF1 StdComment ACF 2 **PicComment** ACF3 OpenPicture picFrame: rect ACF4 ClosePicture ACF 5 KillPicture pic: picHandle ACF 6 DrawPicture myPicture: PicHandle; picFrame: Rect ACF 8 ScalePt ACF 9 MapPt

File:

ACFA

ACFB

MapRect

MapRgn

Report: TrapList

ToolBox Names

File: ToolBox Names Page 6
Report: TrapList July 14 1983

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF

Value/ Name: Fields: ACFC MapPoly InitFonts ACFE ACFF GetFontName AD00 GetFNum AD01 **FMSwapFont** fntRec: FontRec; VAR fontError: Integer AD03 SetFontLock AD06 str: Str40 NewString AD07 SetString sh : StringHandle; str : Str40 AD08 ShowHide AD09 CalcVis window: WindowPeek ADOA CalcVisBehind startWindow: WindowPeek; clobbered: RgnHandle ADOB ClipAbove window: WindowPeek AD0C PaintOne window: WindowPeek; clobbered: RgnHandle ADOD PaintBehind startWindow: WindowPeek; clobbered: RgnHandle AD0E SaveOld window: WindowPeek ADOF DrawNew window: WindowPeek; fUpdate: Boolean AD10 GetWMgrPort AD11 CheckUpDate VAR the Event: EventRecord AD12 InitWindows AD13 NewWindow boundsRect, title, visible, kind, theProc, behind, refCon AD14 theWindow: WindowPtr DisposeWindow theWindow: WindowPtr AD15 ShowWindow AD16 HideWindow theWindow: WindowPtr theWindow: WindowPtr AD17 GetWRefCon AD18 SetWRefCon theWindow: WindowPtr; data: LongInt AD19 GetWTitle theWindow: WindowPtr; VAR Title: Str255 AD1A SetWTitle theWindow: WindowPtr; title: Str255 AD1B MoveWindow theWindow: WindowPtr; hGlobal, vGlobal: Integer AD1C HiliteWindow theWindow: WindowPtr; fHiLite: Boolean AD1D SizeWindow theWindow: WindowPtr; width, height: Integer; fUpdate: Boolean AD1E TrackGoAway AD1F SelectWindow AD20 theWindow: WindowPtr BringToFront theWindow: WindowPtr AD21 SendBehind theWindow: WindowPtr AD22 BeginUpdate AD23 EndUpdate theWindow: WindowPtr AD24 FrontWindow theWindow: WindowPtr; startPt: Point AD25 DragWindow AD26 DragTheRgn AD27 theWindow: WindowPtr; rgn: RgnHandle InvalRgn AD28 InvalRect badRect : Rect AD29 ValidRgn goodRgn : RgnHandle AD2A goodRect : Rect ValidRect AD2B GrowWindow AD2C FindWindow AD2D CloseWindow AD2E SetWindowPic AD2F GetWindowPic

File: ToolBox Names Page 7
Report: TrapList July 14 1983

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Name: Fields:

AD30 InitMenus AD31 NewMenu menuID : INTEGER; menuTitle : Str255 AD32 DisposeMenu menu : MenuHandle AD33 AppendMenu menu : MenuHandle; data : Str255 AD34 ClearMenuBar AD35 InsertMenu menu : menuHandle; beforeId: INTEGER; AD36 DeleteMenu menuId: Integer AD37 DrawMenuBar AD38 HiliteMenu menuId: Integer AD39 EnableItem menu: MenuHandle; item : INTEGER; AD3A DisableItem menu: menuHandle; item: INTEGER AD3B GetMenuBar AD3C SetMenuBar menuBar : Handle AD3D MenuSelect startPt: Point AD3E MenuKey ch: Char AD3F GetItemIcon menu: MenuHandle; item : INTEGER; VAR iconChar : char AD40 SetItemIcon menu: MenuHandle; item : Integer; iconChar : Char AD41 GetItemStyle menu: MenuHandle; item : INTEGER; VAR styleChar : Char AD42 SetItemStyle menu: MenuHandle; item : INTEGER; styleChar : Char AD43 GetItemMark menu: menuHandle; item: INTEGER; VAR markChar: CHAR AD44 SetItemMark menu: menuHandle; item: INTEGER; markChar: CHAR AD45 CheckItem menu : MenuHandle ; item : INTEGER; checked : Boolean AD46 GetItem menu: MenuHandle; item : INTEGER; itemString : Str255 AD47 SetItem menu : MenuHandle; item : INTEGER; itemString: Str255 AD48 CalcMenuSize menu: menuHandle AD49 GetMHandle menuId: INTEGER AD4A SetMenuFlash menu: MenuHandle; flashCount : INTEGER AD4B PlotIcon AD4C FlashMenuBar AD4D AddResMenu theMenu: menuHandle; theType: ResType AD4E PinRect AD4F DeltaPoint AD50 CountItems AD51 InsertResMenu AD54 ownWindow: WindowPtr; boundsRect; title; visible; value; min; NewControl AD55 DisposeControl theControl: ControlHandle AD56 KillControls theWindow: windowPtr AD57 ShowControl theControl : ControlHandle AD58 HideControl theControl : ControlHandle AD59 MoveControl theControl: ControlHandle; h,v: INTEGER AD5A GetCRefCon theControl : ControlHandle AD5B SetCRefCon theControl : ControlHandle; data : LongInt AD5C SizeControl theControl : ControlHandle; w, h : INTEGER AD5D HiliteControl theControl: ControlHandle; hiliteState: INTEGER AD5E GetCTitle theControl : ControlHandle; VAR title : Str255 AD5F SetCTitle theControl: ControlHandle; title: Str255 AD60 GetCValue theControl : ControlHandle AD61 theControl : ControlHandle GetCtlMin

Page 8 July 14 1983

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Fields: Value/ Name: AD62 GetCtlMax theControl : ControlHandle theControl : ControlHandle; theValue : INTEGER AD63 SetCtlValue AD64 SetCtlMin theControl: ControlHandle; theValue: INTEGER SetCtlMax theControl: ControlHandle; theValue: INTEGER AD65 AD66 TestControl theControl: ControlHandle; thePt: Point theControl: ControlHandle; startPt: Point; bounds: Rect; axis: AD67 DragControl theControl: ControlHandle; actionProc: ProcPtr AD68 TrackControl AD69 DrawControls theWindow: WindowPtr AD6A GetCtlAction theControl: ControlHandle SetCtlAction theControl: ControlHandle; actionProc: procPtr AD6B AD6C FindControl the Event: EventRecord; VAR the Window: WindowPtr; VAR the Contro mask: Integer: VAR the Event: EventRecord AD70 GetNextEvent AD71 mask: Integer: VAR the Event: EventRecord EventAvail AD72 GetMouse VAR pt: Point AD73 Still Down AD74 Button AD75 TickCount AD76 GetKeys AD77 WaitMouseUp AD79 FP68K AD7B InitDialogs GetNewDialog AD7C AD7D NewDialog AD7E SetIText AD7F IsDialogEvent AD80 DialogSelect AD81 DrawDialog AD82 CloseDialog AD83 DisposeDialog AD85 Alert AD86 StopAlert alertNo : Integer AD87 NoteAlert alertNo : Integer AD88 CautionAlert alertNo : Integer AD89 CouldAlert alertNo : Integer alertNo : Integer AD8A FreeAlert AD8B ParamText citel, cite2, cite3 : Str20 AD8C ErrorSound AD8D GetDItem AD8E SetDItem AD8F SetIText AD90 GetIText AD91 ModalDialog AD92 DetachResouce AD93 SetResPurge AD94 CurResFile AD95 InitResources

ToolBox Names

File:

AD96

AD97

RsrcZoneInit

OpenResFile

fileName: Str255

Report: TrapList

File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Name: Fields:

Value/ Name: AD98 refnum: INTEGER UseResFile UpdateResFile refnum: INTEGER AD99 CloseResFile refNum : INTEGER AD9A load: BOOLEAN AD9B SetResLoad CountResources theType: ResType AD9C GetIndResource theType: ResType; index: INTEGER AD9D AD9E CountTypes index: INTEGER AD9F GetIndType GetResource theType: ResType; name: Str255 ADA0 GetNamedResourc theType: ResType; name: Str255 ADA1 theResource: Handle ADA2 LoadResource ReleaseResource theResource: Handle ADA3 BeginSubResourc theResource: Handle ADA4 ADA 5 EndSubResource theResource: Handle ADA6 GetResAttrs ADA7 SetResAttrs theResource: Handle; attrs: INTEGER theResource: handle; VAR theId: INTEGER; VAR theType: ResType; ADA8 GetResInfo theResource: Handle; theId: INTEGER; name: Str255 ADA9 SetResInfo ChangedResData theResource: Handle ADAA theData: handle; theType: ResType; theID: INTEGER; thename: St ADAB AddResource theResource: handle; theId: INTEGER; name: Str255 ADAC AddReference ADAD RmveResource theResource: Handle ADAE RmveReference theResource: Handle ADAF ResError ADBO WriteResource ADB1 CreateResFile ADB 2 SystemEvent theEvent: EventRecord ADB 3 SystemClick the Event: EventRecord; the Window: WindowPtr ADB4 SystemTask ADB 5 SystemMenu menuResult: LongInt ADB6 OpenDeskAcc ADB7 CloseDeskAcc ADB8 GetPattern ADB9 GetCursor ADBA GetString ADBB GetIcon ADBC GetPicture ADBD GetNewWindow ADBE GetNewControl ADBF GetMenu ADC 0 GetNewMBar ADC 1 UniqueID ADC 2 SystemEdit ADC 4 LBin2Dec ADC 5 LDec2Bin ADC6 Secs2Date ADC 7 Date2Secs ADC 9 SystemError

File: ToolBox Names Page 10 July 14 1983 Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Name: Fields:

ADF4 ExitToShell ADF5 GetAppParms

varue/	Name:	rieids:
***	20000000000000000000000000000000000000	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADCA	PutIcon	•
ADCC	TEInit	
ADCD	TEDispose	
ADCE	TextBox	
ADCF	TESetText	
ADD0	TECalText	•
ADD1	TESetSelect	•
ADD2	TENew	
ADD3	TEUpdate	
ADD4	TEClick	
ADD5	TECopy	•
ADD6	TECut	•
ADD7	TEDelete	
ADD8	TEActivate	•
ADD9	TEDeactivate	•
ADDA	TEIdle	
ADDB	TEPaste	•
ADDC	TEKey	
ADDD	TEScroll	•
ADEO	Munger	
ADE1	Hand To Hand	
ADE 2	PtrToXHand	•
ADE 3	PtrToHand	•
ADE 5	InitPack	
ADE 6	InitMath	•
ADE 7	Pack0	
ADE8	Packl	
ADE9	Pack2	
ADEA	Pack3	•
ADEB	Pack4	•
ADEC	Pack5	•
ADED	Pack6	•
ADEE	Pack7	•
ADF O	LoadSeg	
ADF1	UnLoadSeg	•
ADF 2	Launch	•
ADF3	Chain	기독이 가면 생각이 되었다. 그는 사람들이 가입니다 가장

ToolBox Names File: July 14 1983 Report: TrapList Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Fields: Name: A04E -AddDrive source(4) dest(4) AC7E AddPt theResource: handle; theId: INTEGER; name: Str255 ADAC AddReference theMenu: menuHandle; theType: ResType AD4D AddResMenu theData: handle; theType: ResType; theID: INTEGER; thename: St ADAB AddResource AD85 Alert AD33 menu : MenuHandle; data : Str255 AppendMenu color(4) AC63 BackColor BackPat AC7C pat(4) BeginSubResourc ADA4 theResource: Handle theWindow: WindowPtr AD22 BeginUpdate long1(4) long2(4) AC58 BitAnd AC5F bytePtr(4) bitNum(4) BitClr AC 5A long(4) BitNot AC5B long1(4) long2(4)BitOr AC5E bytePtr(4) binNum(4) BitSet BitShift AC 5C BitTst AC5D bytePtr(4) bitNum(4) BitXor AC 59 long1(4) long2(4)BlockMove A02E SourcePtr(A0) DestPtr(A1) Count(D0) theWindow: WindowPtr AD20 BringToFront Button AD74 menu: menuHandle CalcMenuSize AD48 CalcVis ADO9 window: WindowPeek CalcVisBehind AD0A startWindow: WindowPeek; clobbered: RgnHandle CautionAlert AD88 alertNo : Integer Chain ADF 3 ChangedResData ADAA theResource: Handle CharWidth AC8D ch(2)CheckItem AD45 menu : MenuHandle ; item : INTEGER; checked : Boolean CheckUpDate AD11 VAR theEvent: EventRecord ClearMenuBar AD34 ClipAbove AD0B window: WindowPeek ClipRect AC7B r(4) Close A001 CloseDeskAcc ADB 7 CloseDialog AD82 ClosePicture ACF 4 ClosePoly ACCC ClosePort AC7D grafPointer(4) CloseResFile AD9A refNum : INTEGER CloseRgn ACDB rgn: RgnHandle CloseWindow AD2D CmpString A03C ColorBit AC64 whichBit(2)CompactMem A04C Control A004

CopyBits

ACEC

Page 2 July 14 1983

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Fields: CopyRgn ACDC srcRgn, dstRgn: RgnHandle AD89 CouldAlert alertNo : Integer CountItems AD50 CountResources AD9C theType: ResType AD9E CountTypes CreateResFile ADB1 CurResFile AD94 ADC 7 Date2Secs Delay A03B DeleteMenu AD36 menuld: Integer DeltaPoint AD4F DetachResouce AD92 DialogSelect AD80 DiffRgn ACE6 srcRgnA, srcRgnB, dstRgn: RgnHandle DisableItem AD3A menu: menuHandle; item: INTEGER DisposeControl AD55 theControl : ControlHandle AD83 DisposeDialog AD32 menu : MenuHandle DisposeMenu A01F Pointer(idA0) DisposePtr DisposeRgn ACD9 rgn: RgnHandle DisposeWindow AD14 theWindow: WindowPtr AD67 theControl: ControlHandle; startPt: Point; bounds: Rect; axis: DragControl AD26 DragTheRgn theWindow: WindowPtr; startPt: Point AD25 DragWindow DrawChar AC83 AD69 theWindow: WindowPtr DrawControls DrawDialog AD81 DrawMenuBar AD37 ADOF window: WindowPeek; fUpdate: Boolean DrawNew myPicture: PicHandle; picFrame: Rect ACF 6 DrawPicture AC84 s(4)DrawString AC85 textBuf(4) firstBute(2) byteCount(2) DrawText DrvrInstall A03D DrvrRemove A03E A023 Handle(A0) DsposeHandle A017 Eject Handle(ihA0) A02B EmptyHandle ACAE EmptyRect ACE2 rgn: RgnHandle EmptyRgn AD39 menu: MenuHandle; item : INTEGER; EnableItem ADA 5 EndSubResource theWindow: WindowPtr AD23 EndUpdate ptA(4) ptB(4) EqualPt AC81 EqualRect ACA6 rgnA rgnB: RgnHandle ACE3 EqualRgn r : Rect; startAngle, arcAngle : INTEGER ACC0 EraseArc EraseOval ACB9 r: Rect poly: PolyHandle ACC8 ErasePoly

File:

Report: TrapList

ToolBox Names

File: ToolBox Names

Report: TrapList

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Name: Value/ Fields:

r: Rect ACA3 EraseRect ACD4 rgn: RgnHandle EraseRgn ACB2 r: Rect: ovalWidth, ovalheight: Integer EraseRoundRect AD8C ErrorSound EventAvail AD71 mask: Integer; VAR the Event: EventRecord ExitToShell ADF 4 A010 FileAllocate FileCreate 800A A009 FileDelete r : Rect; startAngle, arcAngle : INTEGER; pat : pattern FillArc ACC2 r: Rect; pat: Pattern FillOval ACBB FillPoly ACCA poly: PolyHandle; pat: Pattern ACA5 FillRect r: Rect; pat: Pattern FillRgn ACD6 rgn: RgnHandle; pat: Pattern FillRoundRect ACB4 r: Rect; ovalWidth, ovalHeight: Integer; pat: Pattern FindControl AD6C the Event: EventRecord; VAR the Window: WindowPtr; VAR the Contro AD2C FindWindow A016 F Init Queue FixMul AC68 a(4) b(4)FixRatio AC 69 numerator(2) denominator(2) FixRound AC6C x(4)FlashMenuBar AD4C FlushEvents A032 FlushFil A045 FlushVol A013 **FMSwapFont** AD01 fntRec: FontRec; VAR fontError: Integer ForeColor AC62 color(4) FP68K AD79 FrameArc ACBE r : Rect; startAngle, arcAngle : INTEGER r: Rect FrameOval ACB 7 ACC6 FramePoly poly: PolyHandle ACA1 FrameRect r: Rect FrameRgn ACD2 rgn: RgnHandle FrameRoundRect ACB0 r: Rect; ovalWidth, ovalHeight: Integer FreeAlert AD8A alertNo : Integer FreeMem A01C BytesFree(odD0) FrontWindow AD24 GetAppParms ADF 5 GetClip AC7A rgn(4)GetCRefCon AD 5A theControl: ControlHandle GetCTitle AD5E theControl: ControlHandle; VAR title: Str255 GetCtlAction AD6A theControl: ControlHandle GetCtlMax AD62 theControl : ControlHandle GetCtlMin AD61 theControl: ControlHandle GetCursor ADB 9 GetCValue AD60 theControl: ControlHandle GetDItem AD8D GetEOF A011

ToolBox Names File: Page 4 Report: TrapList July 14 1983 Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Fields: Name: GetFileInfo A00C AD00 GetFNum GetFontInfo AC8B info(4) GetFontName ACFF GetFPos A018 GetHandleSize A025 Handle(A0) Size(D0) GetIcon ADBB AD9D GetIndResource theType: ResType; index: INTEGER GetIndType AD9F index: INTEGER AD46 menu: MenuHandle; item : INTEGER; itemString : Str255 GetItem AD3F menu: MenuHandle; item : INTEGER; VAR iconChar : char GetItemIcon AD43 menu: menuHandle; item: INTEGER; VAR markChar: CHAR GetItemMark AD41 menu: MenuHandle; item : INTEGER; VAR styleChar : Char GetItemStyle AD90 GetIText AD76 GetKeys ADBF GetMenu GetMenuBar AD3B menuId: INTEGER GetMHandle AD49 AD72 GetMouse VAR pt: Point the Type: Res Type; name: Str255 GetNamedResourc ADA1 GetNewControl ADBE AD7C GetNewDialog ADCO GetNewMBar GetNewWindow ADBD GetNextEvent AD70 mask: Integer; VAR the Event: EventRecord GetOSEvent A031 GetPattern ADB8 AC9A pt(4) GetPen GetPenState AC98 pnState(4) ADBC GetPicture GetPixel AC65 h(2) v(2)AC74 grafPtr(4) GetPort GetPtrSize A021 Pointer(idA0) Size(odD0) GetResAttrs ADA 6 theResource: Handle theResource: handle; VAR theId: INTEGER; VAR theType: ResType; GetResInfo ADA8 ADA 0 theType: ResType; name: Str255 GetResource GetString ADBA GetTrapAddress A046 A014 GetVol A007 GetVolInfo AD2F GetWindowPic GetWMgrPort AD10 GetWRefCon AD17 theWindow: WindowPtr GetWTitle AD19 theWindow: WindowPtr; VAR Title: Str255 GetZone A01A theZone(odA0)

AC71

AC72

AD2B

pt(4)

device(2)

GlobalToLocal

GrafDevice

GrowWindow

July 14 1983 Report: TrapList Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Fields: Name: A026 Handle(A0) HeapPtr(A0) HandleZone ADE 1 **HandToHand** theControl: ControlHandle HideControl AD58 AC52 HideCursor AC96 HidePen theWindow: WindowPtr HideWindow AD16 theControl: ControlHandle; hiliteState: INTEGER HiliteControl AD5D AD38 menuId: Integer HiliteMenu HiliteWindow AD1C theWindow: WindowPtr; fHiLite: Boolean HiWord AC 6A x(4)A029 Handle(A0) HLock HNoPurge AO4A Handle(A0) HPurge A049 Handle(A0) HUnlock A02A Handle(A0) InitApplZone A02C AC 50 InitCursor InitDialogs AD7B InitFonts ACFE InitGraf AC6E globalPtr(4) InitMath ADE 6 InitMenus AD30 InitPack ADE 5 InitPort AC6D grafpointer(4) InitResources AD95 InitUtil A03F InitWindows AD12 InitZone A019 startPtr(id4) limitPtr(id4) masterCount(id2) growZone(ipc4) InsertMenu AD35 menu : menuHandle; beforeId: INTEGER; InsertResMenu AD51 InsetRect ACA9 VAR r: Rect; dh, dv: Integer ACE1 InsetRgn rgn: RgnHandle; dh, dv: Integer InstallDrivers A04F InvalRect AD28 badRect : Rect InvalRgn AD27 theWindow: WindowPtr; rgn: RgnHandle InvertArc ACC1 r : Rect; startAngle, arcAngle : INTEGER Invert0val ACBA r: Rect InvertPoly ACC9 poly: PolyHandle InvertRect ACA4 r: Rect InvertRgn ACD5 rgn: RgnHandle InvertRoundRect ACB3 r: Rect; ovalWidth, ovalHeight: Integer IsDialogEvent AD7F KillControls AD56 theWindow: windowPtr KillIO A006 KillPicture ACF 5 pic: picHandle KillPoly ACCD poly: PolyHandle Launch ADF 2 LBin2Dec ADC 4

ToolBox Names

File:

LDec2Bin

ADC 5

File: ToolBox Names
Report: TrapList
Selection: Value/Trap: equals A000
through Value/Trap: equals AFFF
Name: Value/ Fields:

ADEA

Pack3

```
AC92
                        dh(2) dv(2)
Line
                 AC91
                        h(2) v(2)
LineTo
LoadResource
                 ADA 2
                        theResource: Handle
                 ADF 0
LoadSeg
LocalToGlobal
                 AC70
                        pt(4)
                        a(4) b(4) dst(4)
                 AC67
Long Mul
                 AC6B
                        \mathbf{x}(4)
LoWord
                 ACFC
MapPoly
                 ACF9
MapPt
                 ACFA
MapRect
                 ACFB
MapRgn
                        MaxBytes(odD0)
                 A01D
MaxMem
                        ch: Char
                 AD3E
MenuKey
                        startPt: Point
                 AD3D
MenuSelect
ModalDialog
                 AD91
                 AC94
                        dh(2) dv(2)
Moove
                 AOOF
MountVol
                        theControl: ControlHandle; h,v: INTEGER
                 AD59
MoveControl
                        leftGlobal(2) topGlobal(2)
                 AC77
MovePort
                        h(2) v(2)
                 AC93
MoveTo
                        theWindow: WindowPtr; hGlobal, vGlobal: Integer
MoveWindow
                 AD1B
                 ADE 0
Munger
                        ownWindow: WindowPtr; boundsRect; title; visible; value; min;
                 AD54
NewControl
                 AD7D
NewDialog
                        BytesNeeded(idD0) Handle(odA0)
                 A022
NewHandle
                         menuID : INTEGER; menuTitle : Str255
                 AD31
NewMenu
                         byteCount(idD0) Pointer(odA0)
                 A01E
NewPtr
                 ACD8
NewRgn
                         str: Str40
                 AD06
NewString
                         boundsRect, title, visible, kind, the Proc, behind, refCon
                 AD13
NewWindow
                 AD87
                         alertNo : Integer
NoteAlert
                 AC56
ObscureCursor
                         poly: PolyHandle; dh, dv: Integer
                 ACCE
OffsetPoly
                         VAR r: Rect; dh, dv: Integer
OffsetRect
                 ACA8
                         rgn: RgnHandle; dh, dv: Integer
                 ACE0
OffsetRgn
                 A000
Open
                 ADB 6
OpenDeskAcc
                         picFrame: rect
                 ACF 3
OpenPicture
                  ACCB
 OpenPoly
                         grafPointer(4)
                  AC6F
 OpenPort
                  AD97
                         fileName: Str255
 OpenResFile
                  A00A
 OpenRf
                  ACDA
 OpenRgn
                  A030
 OSEventAvail
                  ADE 7
 Pack0
                  ADE8
 Packl
                  ADE9
 Pack2
```

File: ToolBox Names July 14 1983 Report: TrapList Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Fields: Name: ADEB Pack4 ADEC Pack5 ADED Pack6 Pack7 ADEE PackBits ACCF r : Rect; startAngle, arcAngle : INTEGER ACBF PaintArc PaintBehind ADOD startWindow: WindowPeek; clobbered: RgnHandle AD0C window: WindowPeek; clobbered: RgnHandle PaintOne ACB8 PaintOval r: Rect PaintPoly ACC7 poly: PolyHandle PaintRect ACA2 r: Rect rgn: RgnHandle PaintRgn ACD3 PaintRoundRect ACB1 r: Rect; ovalWidth, ovalheight: Integer AD8B ParamText citel, cite2, cite3 : Str20 PenMode AC9C mode(2)PenNormal AC9E PenPat AC9D pat(4) PenSize AC9B width(2) height(2) Pic Comment ACF 2 PinRect AD4E PlotIcon AD4B PortSize AC76 width(2) height(2) PostEvent A02F Pt2Rect ACAC ptl, pt2: Point: VAR dst: Rect Pt InRect ACAD pt: Point; r: Rect ACE8 Pt InRgn pt: Point; rgn: RgnHandle PtrToHand ADE 3 PtrToXHand ADE 2 PtrZone A048 theZone(ihA0) Pt To Angle ACC3 r : Rect; pt : Point; VAR angle : INTEGER PurgeMem AO4D Put Icon ADCA Random AC61 Read A002 ReadDateTime A039 ReadParam A037 ReAllocHandle A027 Handle(A0) Size(D0) RecoverHandle A028 Pointer(A0) Handle(A0) RectInRgn ACE9 r:Rect; rgn: RgnHandle RectRgn ACDF rgn : RgnHandle; r : Rect ReleaseResource ADA3 theResource: Handle Rename A00B ResError ADAF RmveReference ADAE theResource: Handle RmveResource ADAD theResource: Handle RsrcZoneInit AD96 RstFilLock A042 SaveOld AD0E window: WindowPeek

File: ToolBox Names Page 8
Report: TrapList July 14 1983

Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Name: Value/ Fields:

```
ACF 8
ScalePt
                ACEF
                        dstRect: Rect; dh, dv: Integer; updateRgn: rgnHandle
ScrollRect
Secs2Date
                ADC 6
                        srcl, src2: Rect; VAR dst: Rect
SectRect
                ACAA
SectRgn
                ACE4
                        srcRgnA, srcRgnB, dstRgn: RgnHandle
                AD1F
SelectWindow
                AD21
                        theWindow: WindowPtr
SendBehind
                AC57
SetApplBase
                A02D
SetApplLimit
                AC79
                        rgn(4)
SetClip
                        theControl: ControlHandle; data: LongInt
SetCRefCon
                AD5B
SetCTitle
                AD5F
                        theControl: ControlHandle; title: Str255
                AD6B
                        theControl: ControlHandle; actionProc: procPtr
SetCtlAction
                        theControl: ControlHandle; theValue: INTEGER
SetCtlMax
                AD65
                        theControl: ControlHandle; theValue: INTEGER
                AD64
SetCtlMin
                        theControl: ControlHandle; theValue: INTEGER
SetCtlValue
                AD63
                AC 51
                        cursor(4)
SetCursor
                AO3A
SetDateTime
                AD8E
SetDItem
                ACDD
SetEmptyRgn
                A012
SetEOF
                 COOA
SetFileInfo
                A041
SetFilLock
SetFilType
                 A043
                 ADO3
SetFontLock
                 A044
SetFPos
                 AO4B
SetGrowZone
                        Handle(A0) Size(D0)
                 A024
SetHandleSize
                        menu : MenuHandle; item : INTEGER; itemString: Str255
                 AD47
SetItem
                        menu: MenuHandle; item : Integer; iconChar : Char
SetItemIcon
                 AD40
                        menu: menuHandle; item: INTEGER; markChar: CHAR
                 AD44
SetItemMark
                        menu: MenuHandle; item : INTEGER; styleChar : Char
SetItemStyle
                 AD42
                AD7E
SetIText
                 AD8F
SetIText
                        menuBar : Handle
                 AD3C
SetMenuBar
                        menu: MenuHandle; flashCount : INTEGER
SetMenuFlash
                 AD4A
                 AC78
                        h(2) v(2)
SetOrigin
SetPenState
                 AC99
                        pnState(4)
                 AC73
                        grafPtr(4)
SetPort
                 AC75
                        the BitMap(4)
SetPortBits
                        pt(4) h(2) v(2)
                 AC80
SetPt
                        Pointer(idA0) Size(idD0)
                 A020
SetPtrSize
                        VAR r: Rect; left, top, right, bottom: Integer
                 ACA7
SetRect
                        rgn: RgnHandle; left, top, right, bottom: Integer;
                 ACDE
SetRectRgn
                        theResource: Handle; attrs: INTEGER
                 ADA 7
SetResAttrs
                        theResource: Handle; theId: INTEGER; name: Str255
                 ADA 9
SetResInfo
                 AD9B
                        load: BOOLEAN
SetResLoad
                 AD93
SetResPurge
```

Page 9

File: ToolBox Names July 14 1983 Report: TrapList Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF Value/ Fields: SetStdProcs ACEA AD07 SetString sh : StringHandle; str : Str40 SetTrapAddress A047 SetVol A015 AD2E SetWindowPic SetWRefCon AD18 theWindow: WindowPtr; data: LongInt SetWTitle AD1A theWindow: WindowPtr; title: Str255 SetZone A01B theZone(idA0) ShieldCursor AC55 AD57 theControl : ControlHandle ShowControl ShowCursor AC53 ShowHide AD08 AC 97 ShowPen ShowWindow AD15 theWindow: WindowPtr SizeControl AD5C the Control: ControlHandle; w, h: INTEGER SizeWindow AD1D theWindow: WindowPtr; width, height: Integer; fUpdate: Boolean SpaceExtra AC8E extra(2)A005 Status StdArc verb : GrafVerb ; r : Rect ; startAngle, arcAngle : INTEGER ACBD StdBits ACEB VAR scrBits : BitMap; VAR srcRect, dstRect : Rect; mode : Inte ACF1 StdComment StdGetPic ACEE StdLine AC 90 newPt(4) Std0val ACB6 verb : GrafVerb ; r : Rect StdPoly ACC 5 verb : GrafVerb ; poly : PolyHandle StdPutPic ACF 0 StdRect ACA0 verb : GrafVerb ; r : Rect StdRgn ACD1 verb : GrafVerb ; rgn : RgnHandle StdRRect ACAF verb : GrafVerb ; r : Rect; ovW, ovH : INTEGER StdText AC82 count(2) textAddr(4) StdTxMeasure ACED StillDown AD73 StopAlert AD86 alertNo : Integer StringWidth AC8C s(4)StuffHex AC66 thingPtr(4) s(2)SubPt AC7F source(4) dest(4) SystemClick ADB 3 the Event: EventRecord; the Window: WindowPtr SystemEdit ADC 2 SystemError ADC 9 SystemEvent ADB 2 the Event: Event Record SystemMenu ADB 5 menuResult: LongInt SystemTask ADB4 TEActivate ADD8 TECalText ADD0 TEClick ADD4

TECopy

TEDeactivate

TECut

ADD5

ADD6

ADD9

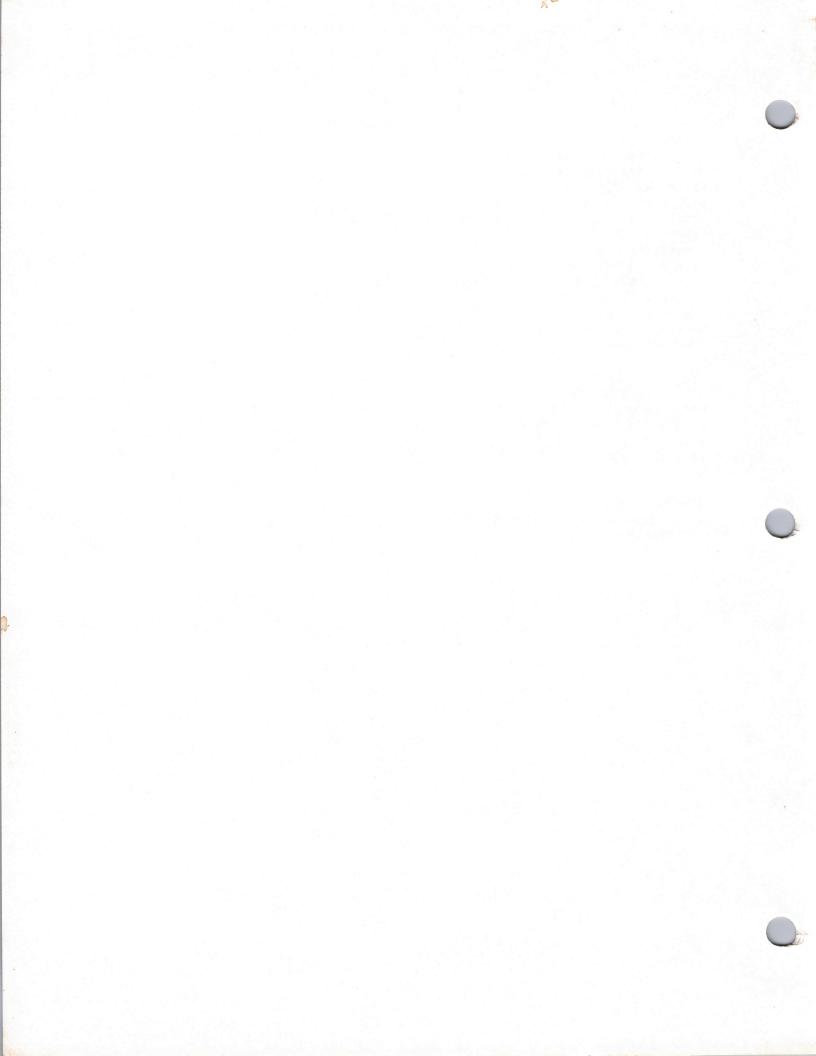
File: ToolBox Names July 14 1983 Report: TrapList Selection: Value/Trap: equals A000 through Value/Trap: equals AFFF

```
Value/ Fields:
Name:
€<br/>
<br/>
<br/
                                                     ADD7
TEDelete
                                                     ADCD
TEDispose
                                                     ADDA
TEIdle
TEInit
                                                     ADCC
                                                     ADDC
TEKey
                                                     ADD2
TENew
                                                     ADDB
TEPaste
                                                     ADDD
TEScrol1
                                                     ADD1
TESetSelect
                                                     ADCF
TESetText
                                                                             theControl : ControlHandle; thePt : Point
TestControl
                                                     AD66
                                                     ADD3
TEUpdate
                                                     ADCE
TextBox
                                                     AC88
                                                                             face(2)
TextFace
                                                                             font(2)
                                                     AC87
TextFont
                                                     AC89
                                                                             mode(2)
TextMode
                                                                             size(2)
                                                     AC8A
TextSize
                                                                              textBuf(4) firstByte(2) byteCount(2)
                                                      AC86
TextWidth
                                                     AD75
 TickCount
                                                                              theControl : ControlHandle; actionProc : ProcPtr
                                                     AD68
 TrackControl
                                                     AD1E
 TrackGoAway
                                                                              srcl, src2: Rect; VAR dst: Rect
                                                      ACAB
 UnionRect
                                                                              srcRgnA, srcRgnB, dstRgn: RgnHandle
                                                      ACE 5
 UnionRen
                                                      ADC 1
 UniqueID
                                                      ADF 1
 UnLoad Seg
                                                      A00E
 Unmount Vol
                                                      ACD0
 UnPackBits
                                                      AD99
                                                                              refnum: INTEGER
 UpdateResFile
                                                      AC54
 UprString
                                                                              refnum: INTEGER
 UseResFile
                                                      AD98
                                                                              goodRect : Rect
                                                      AD2A
 ValidRect
                                                      AD29
                                                                              goodRgn : RgnHandle
 ValidRgn
                                                      A033
 VInstall
                                                       A034
 VRemove
                                                      AD77
 WaitMouseUp
                                                       A003
 Write
                                                       A038
  WriteParam
 WriteResource
                                                       ADB 0
                                                                              srcRgnA, srcRgnB, dstRgn: RgnHandle
                                                       ACE7
  XOrRgn
```

replacing 33, 54
selecting 47, 48
text format 68, 69, 70, 71
title bar 9
type size 90, 91, 96
changing 24, 62
preset 91, 96
removing 62
type style 90, 96
changing 24, 62
preset 91, 96
removing 62
typeface 27, 63
preset 63

Underline command 24
Undo command 19, 89
Undo Ruler Change command 19
Undo Typing command 12
up arrow 18

windows 82 word wraparound 10, 11, 12, 32, 8 wristwatch 8



STRUCTURE APP.

MACINTOSH USER EDUCATION

The Structure of a Macintosh Application

/STRUCTURE/STRUCT

See Also: Macintosh User Interface Guidelines

Inside Macintosh: A Road Map

The Segment Loader: A Programmer's Guide Putting Together a Macintosh Application

Modification History: First Draft (ROM 7)

Caroline Rose

2/8/84

ABSTRACT

This manual describes the overall structure of a Macintosh application program, including its interface with the Finder.

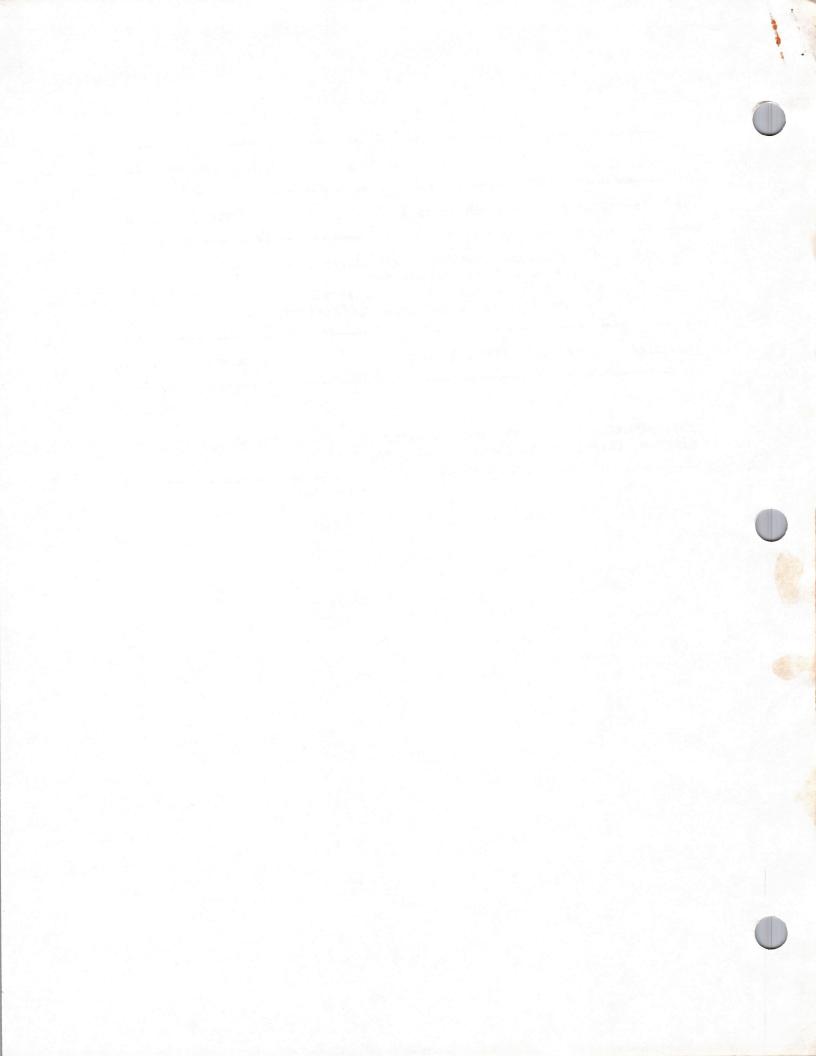
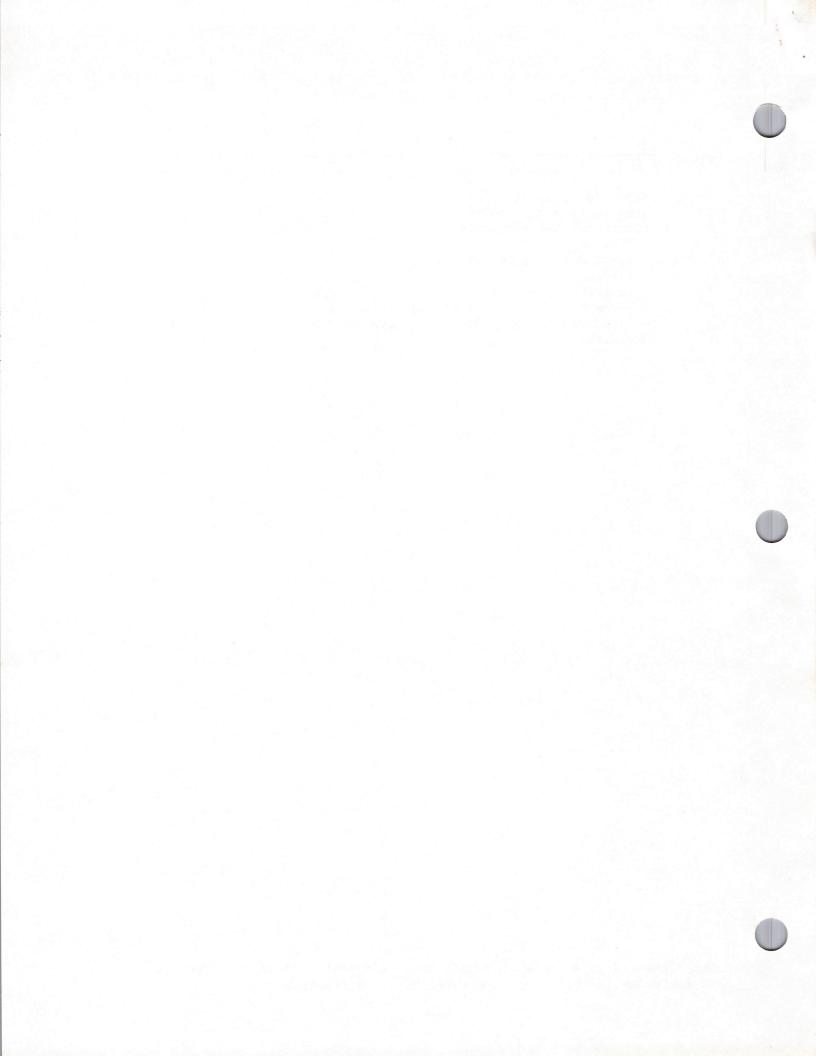


TABLE OF CONTENTS

3	About This Manual
3	Signatures and File Types
4	Finder-Related Resources
5	Version Data
5	Icons and File References
6	Bundles
7	An Example
8	Formats of Finder-Related Resources
8	Opening and Printing Documents from the Finder
11	Glossary



ABOUT THIS MANUAL

This manual describes the overall structure of a Macintosh application program, including its interface with the Finder. *** Right now it describes only the Finder interface; the rest will be filled in later. Eventually it will become part of a comprehensive manual describing the entire Toolbox and Operating System. ***

(hand)

This information in this manual applies to version 7 of the Macintosh ROM and version $1.0\,$ of the Finder.

You should already be familiar with the following:

- The details of the User Interface Toolbox, the Macintosh Operating System, and the other routines that your application program may call. For a list of all the technical documentation that provides these details, see Inside Macintosh: A Road Map.
- The Finder, which is described in the Macintosh owner's guide.

This manual doesn't cover the steps necessary to create an application's resources or to compile, link, and execute the application program. These are discussed in the manual <u>Putting Together a Macintosh Application</u>.

The manual begins with sections that describe the Finder interface: signatures and file types, used for identification purposes; application resources that provide icon and file information to the Finder; and the mechanism that allows documents to be opened or printed from the Finder.

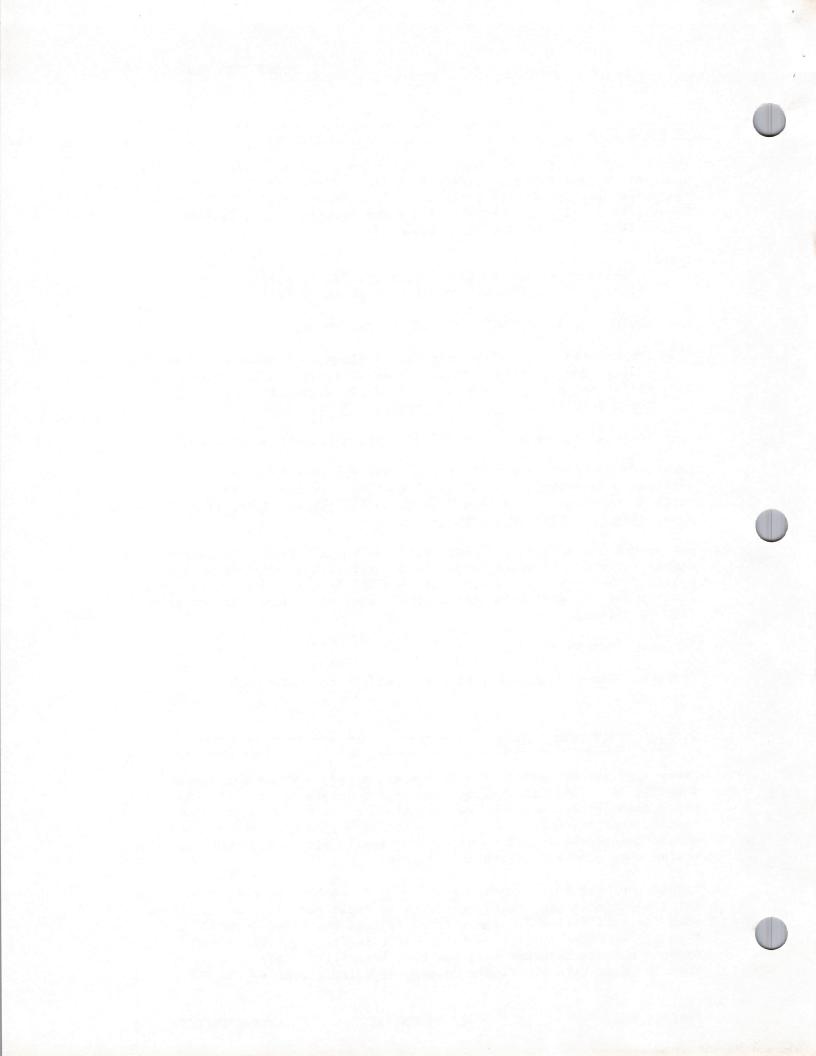
*** more to come ***

Finally, there's a glossary of terms used in this manual.

SIGNATURES AND FILE TYPES

Every application must have a unique <u>signature</u> by which the Finder can identify it. The signature can be any four-character sequence not being used for another application on any currently mounted volume (except that it can't be one of the standard resource types). To ensure uniqueness on all volumes, your application's signature must be assigned by Macintosh Technical Support.

Signatures work together with <u>file types</u> to enable the user to open or print a document (any file created by an application) from the Finder. When the application creates a file, it sets the file's creator and file type. Normally it sets the creator to its signature and the file type to a four-character sequence that identifies files of that type. When the user asks the Finder to open or print the file, the Finder



4 Structure of a Macintosh Application

starts up the application whose signature is the file's creator and passes the file type to the application along with other identifying information, such as the file name. (More information about this process is given below under "Opening and Printing Documents from the Finder".)

An application may create its own special type or types of files. Like signatures, file types must be assigned by Macintosh Technical Support to ensure uniqueness. When the user chooses Open from an application's File menu, the application will display (via the Standard File Package) the names of all files of a given type or types, regardless of which application created the files. Having a unique file type for your application's special files ensures that only the names of those files will be displayed for opening.

(hand)

Signatures and file types may be strange, unreadable combinations of characters; they're never seen by end users of Macintosh.

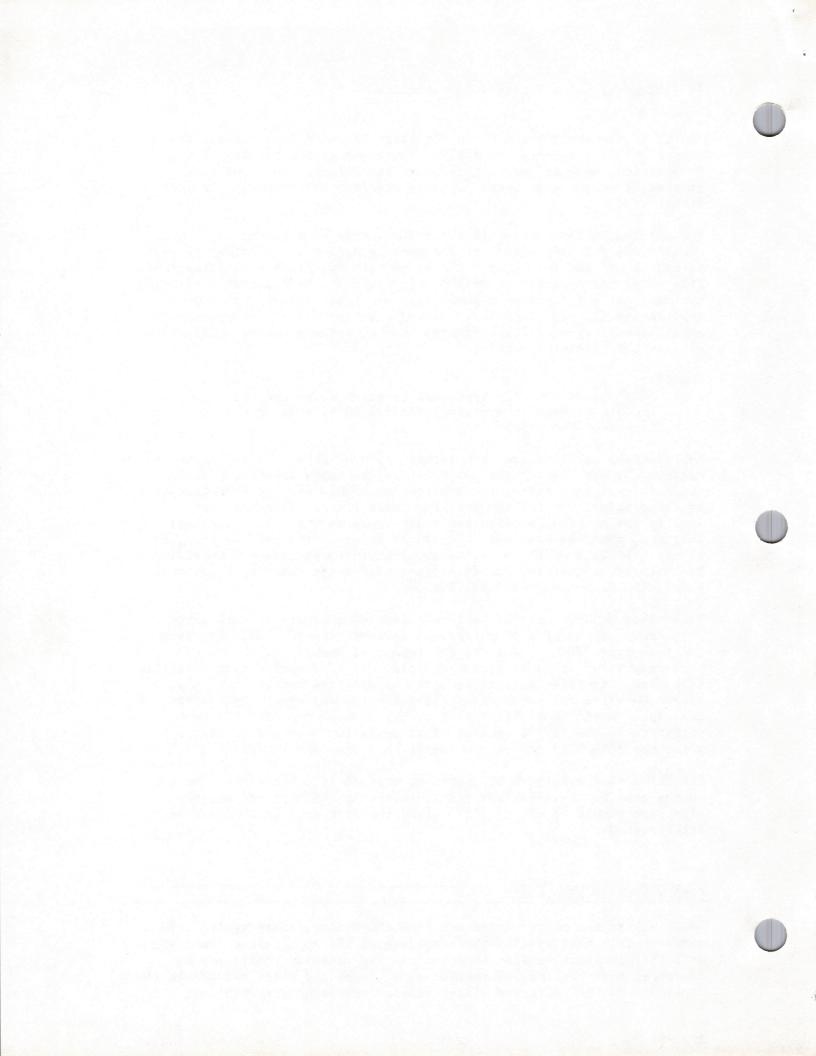
Applications may also create existing types of files. There might, for example, be one that merges two MacWrite documents into a single document. In such cases, the application should use the same file type as the original application uses for those files. It should also specify the original application's signature as the file's creator; that way, when the user asks the Finder to open or print the file, the Finder will call on the original application to perform the operation. To learn the signatures and file types used by existing applications, check with Macintosh Technical Support.

Files that consist only of text—a stream of characters, with Return characters at the ends of paragraphs or short lines—should be given the file type 'TEXT'. This is the type that MacWrite gives to text—only files it creates, for example. If your application uses this file type, its files will be accepted by MacWrite and it in turn will accept MacWrite text—only files (likewise for any other application—that deals with 'TEXT' files). Your application can give its own signature as the file's creator if it wants to be called to open or print the file when the user requests this from the Finder.

For files that aren't to be opened or printed from the Finder, as may be the case for certain data files created by the application, the signature should be set to '????' (and the file type to whatever is appropriate).

FINDER-RELATED RESOURCES

To establish the proper interface with the Finder, every application's resource file must specify the signature of the application along with data that provides version information. In addition, there may be resources that provide information about icons and files related to the application. All of these Finder-related resources are described



below, followed by a comprehensive example and (for interested programmers) the exact formats of the resources.

Version Data

Your application's resource file must contain a special resource that has the signature of the application as its resource type. This resource is called the <u>version data</u> of the application. The version data is typically a string that gives the name, version number, and date of the application, but it can in fact be any data at all. The resource ID of the version data is 9 by convention.

As described in detail in <u>Putting Together a Macintosh Application</u>, part of the process of installing an application on the Macintosh is to set the creator of the file that contains the application. You set the creator to the application's signature, and the Finder copies the corresponding version data into a resource file named <u>Desktop</u>. (The Finder doesn't display this file on the Macintosh desktop, to ensure that the user won't tamper with it.)

(hand)

Additional, related resources may be copied into the Desktop file; see "Bundles" below for more information. The Desktop file also contains folder resources, one for each folder on the volume.

Icons and File References

For each application, the Finder needs to know:

- The icon to be displayed for the application on the desktop, if different from the Finder's default icon for applications (see Figure 1).
- If the application creates any files, the icon to be displayed for each type of file it creates, if different from the Finder's default icon for documents.
- What files, if any, must accompany the application when it's transferred to another volume.



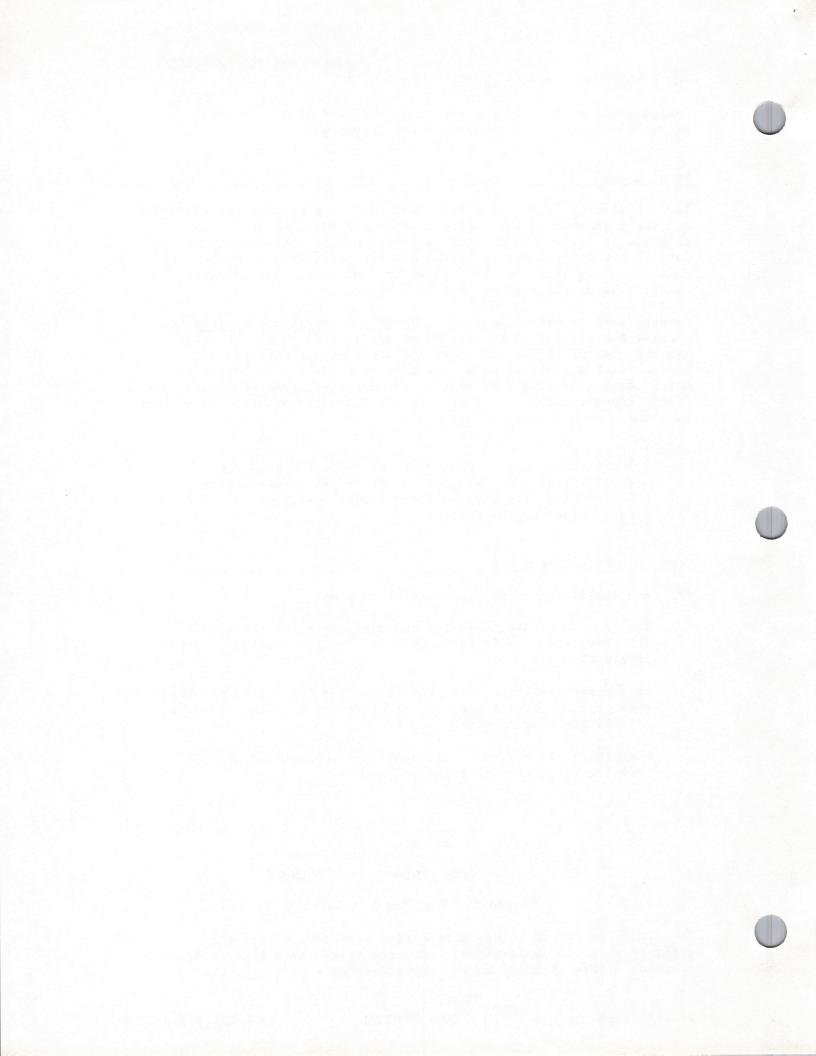
Application



Documen'

Figure 1. The Finder's Default Icons

The Finder learns this information from resources called <u>file</u> references in the application's resource file. Each file reference contains a file type and an ID number, called a <u>local ID</u>, that



6 Structure of a Macintosh Application

identifies the icon to be displayed for that type of file. (The local ID is mapped to an actual resource ID as described under "Bundles" below.) Any file reference may also include the name of a file that must accompany the application when it's transferred to another volume.

The file type for the application itself is 'APPL'. This is the file type in the file reference that designates the application's icon. You also specify it as the application's file type at the same time that you specify its creator—the first time you install the application on the Macintosh.

The ID number in a file reference corresponds not to a single icon but to an <u>icon list</u> in the application's resource file. The icon list consists of two icons: the actual icon to be displayed on the desktop, and a mask consisting of that icon's outline filled with black (see Figure 2). *** For existing types of files, there's currently no way to direct the Finder to use the original application's icon for that file type. ***



Figure 2. Icon and Mask

Bundles

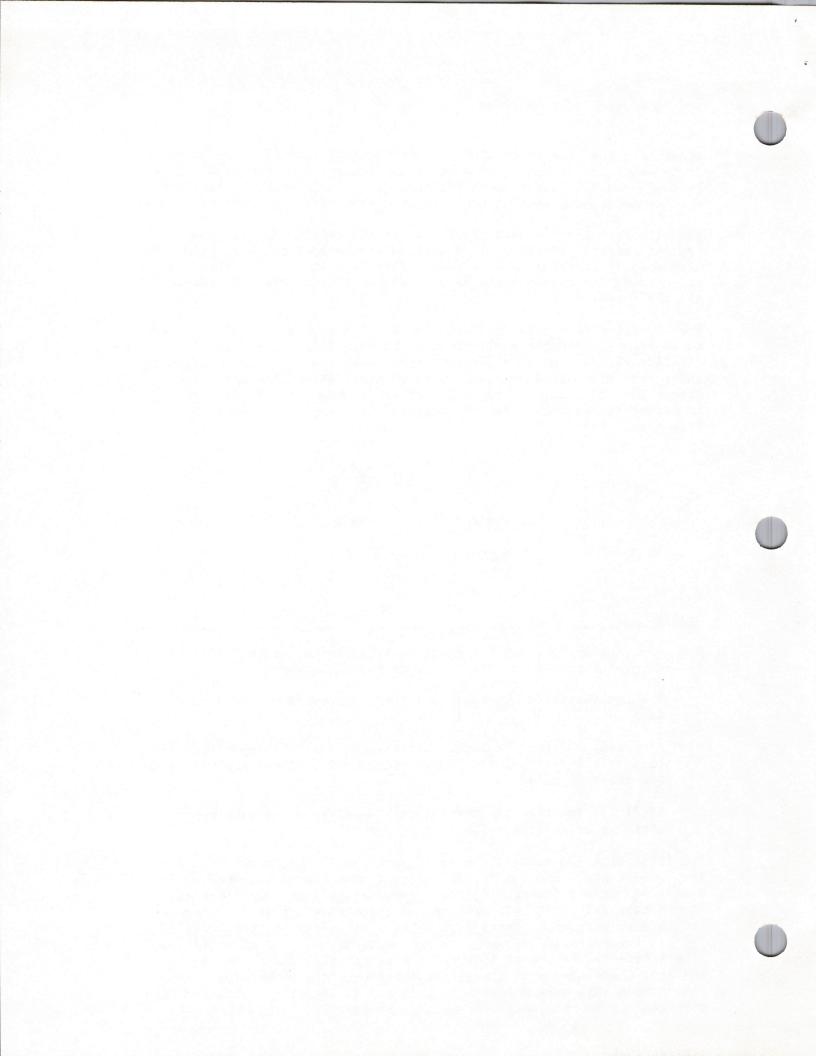
A <u>bundle</u> in the application's resource file groups together all the Finder-related resources. It specifies the following:

- The application's signature and the resource ID of its version data
- A mapping between the local IDs for icon lists (as specified in file references) and the actual resource IDs of the icon lists in the resource file
- Local IDs for the file references themselves and a mapping to their actual resource IDs

The first time you install the application on the Macintosh, you set its "bundle bit", and the Finder copies the version data, bundle, icon lists, and file references from the application's resource file into the Desktop file. *** (The setting of the bundle bit will be covered in the next version of Putting Together a Macintosh Application.) *** If there are any resource ID conflicts between the icon lists and file references in the application's resource file and those in Desktop, the Finder will change those resource IDs in Desktop. The Finder does this same resource copying and ID conflict resolution when you transfer an application to another volume.

2/8//8/ Page

COMPTERMENT



(hand)

The local IDs are needed only for use by the Finder.

An Example

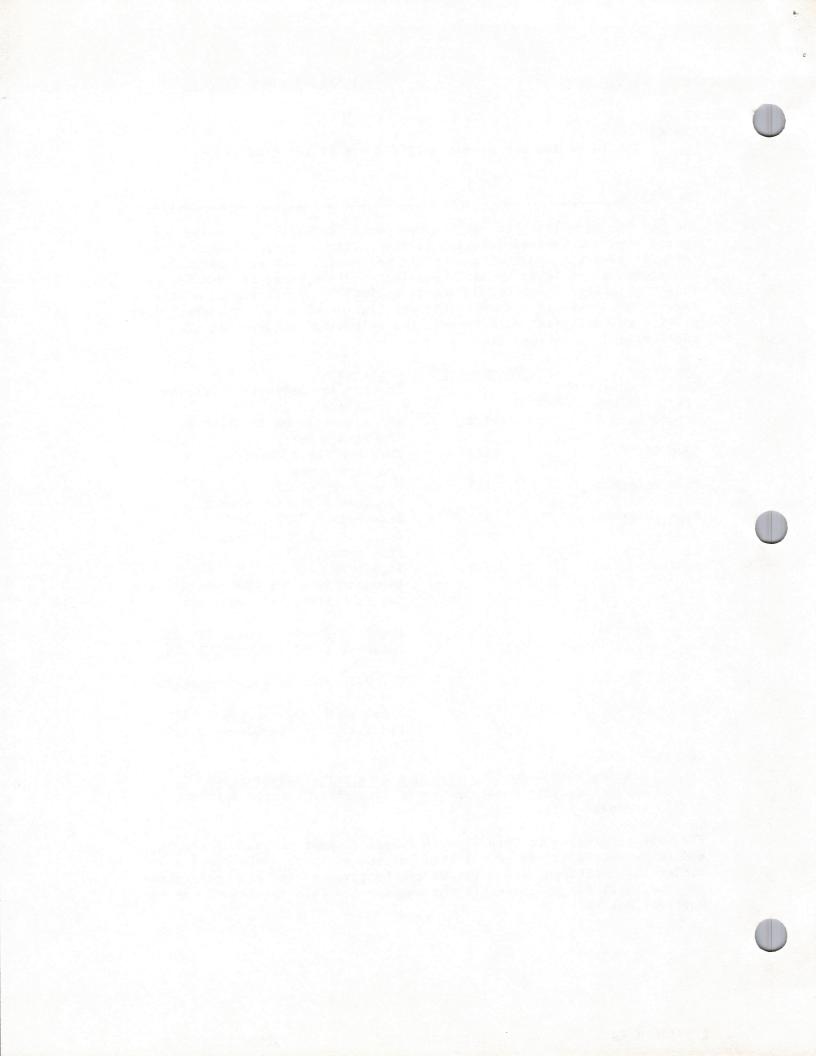
Suppose you've written an application named SampWriter. The user can create a unique type of document from it, and you want a distinctive icon for both the application and its documents. The application's signature, as assigned by Macintosh Technical Support, is 'SAMP'; the file type assigned for its documents is 'SAMF'. Furthermore, a file named 'TgFil' should accompany the application when it's transferred to another volume. You would include the following resources in the application's resource file:

[18] 전환 [18] [18] [18] [18] [18] [18] [18] [18]		에 대한 · · · · · · · · · · · · · · · · · ·
Resource	Resource ID	Contents
Version data with	Ø	The string 'SampWriter Version 1
resource type 'SAMP'		- 2/1/84'
Icon list	128	The icon for the application
Icon list	100	The icon's mask
ICON TISE	129	The icon for documents
T/1		The icon's mask
File reference	128	File type 'APPL'
		Local ID Ø for the icon list
File reference	129	File type 'SAMF'
		Local ID 1 for the icon list
		File name 'TgFil'
Bundle	128	Signature 'SAMP'
		Resource ID Ø for the version data
		For icon lists, the mapping:
		local ID Ø> resource ID 128
		local ID 1> resource ID 129
		For file references, the mapping:
		local ID Ø> resource ID 128
		local ID 1> resource ID 129

(hand)

See the manual <u>Putting Together a Macintosh Application</u> for information about how to include these resources in a resource file.

The file references in this example happen to have the same local IDs and resource IDs as the icon lists, but any of these numbers can be different. Different resource IDs can be given to the file references, and the local IDs specified in the mapping for file references can be whatever desired.



Formats of Finder-Related Resources

The resource type for an application's version data is the signature of the application, and the resource ID is Ø by convention. The resource data can be anything at all; typically it's a string giving the name, version number, and date of the application.

The resource type for an icon list is 'ICN#'. The resource data simply consists of the icons, 128 bytes each.

The resource type for a file reference is 'FREF'. The resource data has the format shown below.

Number of bytes	Contents
4 bytes	File type
2 bytes	Local ID for icon list
l byte	Length of following file name in bytes; Ø if none
n bytes	Optional file name

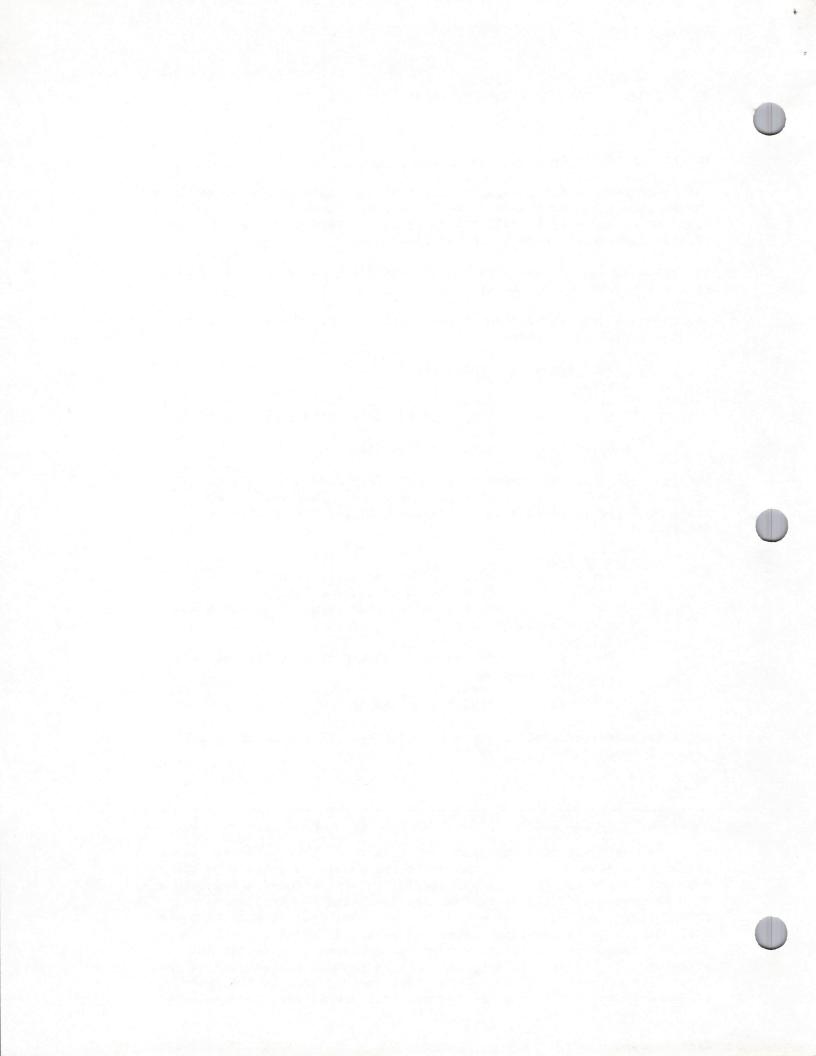
The resource type for a bundle is 'BNDL'. The resource data has the format shown below. The format is more general than needed for Finder-related purposes because bundles will be used in other ways in the future.

Number of bytes	Contents
4 bytes	Signature of the application
2 bytes	Resource ID of version data
2 bytes	Number of resource types in bundle minus l
For each resou	irce type:
4 bytes	Resource type
2 bytes	Number of resources of this type minus 1
For each res	source:
2 bytes	Local ID
2 bytes	Actual resource ID

A bundle used for establishing the Finder interface contains the two resource types 'ICN#' and 'FREF'.

OPENING AND PRINTING DOCUMENTS FROM THE FINDER

When the user selects a document and tries to open or print it from the Finder, the Finder starts up the application whose signature is the document file's creator. An application may be selected along with one or more documents for opening (but not printing); in this case, the Finder starts up that application. If the user selects more than one document for opening without selecting an application, the files must have the same creator. If more than one document is selected for printing, the Finder starts up the application whose signature is the first file's creator (that is, the first one selected if they were selected by Shift-clicking, or the top left one if they were selected



by dragging a rectangle around them).

Any time the Finder starts up an application, it passes along information via the "Finder information handle" in the application parameter area (as described in the Segment Loader manual). Pascal programmers can call the Segment Loader procedure GetAppParms to get the Finder information handle. For example, if applParam is declared as type Handle, the call

GetAppParms(applName, applRefNum, applParam)

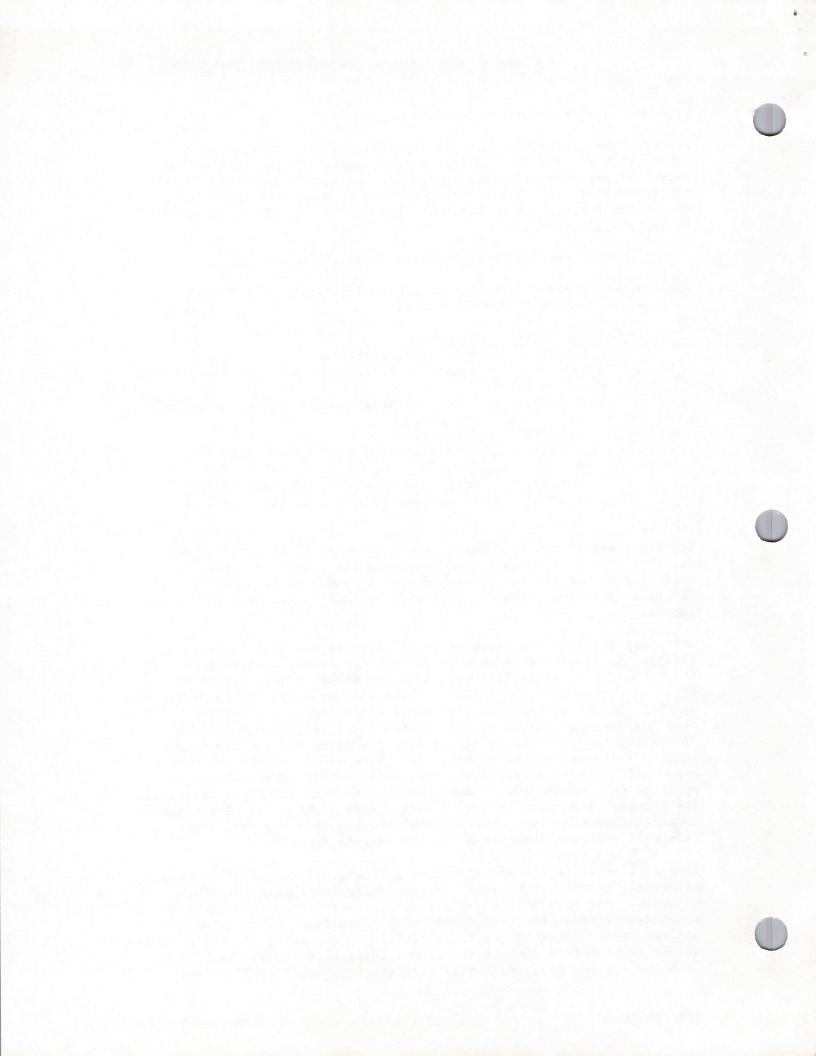
returns the Finder information handle in applParam. The Finder information has the following format:

2 bytes Ø if open, l if print 2 bytes Number of files to open or print (Ø if	none)
	none)
For each file:	
2 bytes Volume reference number of volume conta the file	ining
4 bytes File type	
l byte File's version number (typically 0)	
l byte Ignored	
l byte Length of following file name in bytes	
n bytes Characters of file name (if n is even, an extra byte)	add

The files are listed in order of the appearance of their icons on the desktop, from left to right and top to bottom. The file names don't include a volume prefix. An extra byte is added to any name of even length so that the entry for the next name will begin on a word boundary.

Every application that opens or prints documents should look at this information to determine what to do when the Finder starts it up. If the number of files is Ø, the application should start up with an untitled document on the desktop. If a file or files are specified for opening, it should start up with those documents on the desktop. If only one document can be open at a time but more than one file is specified, the application should open the first one and ignore the rest. If the application doesn't recognize a file's type (which can happen if the user selected the application along with another application's document), it may want to open the file anyway and check its internal structure to see if it's a compatible type. The response to an unacceptable type of file should be an alert box that shows the file name and says that the document can't be opened.

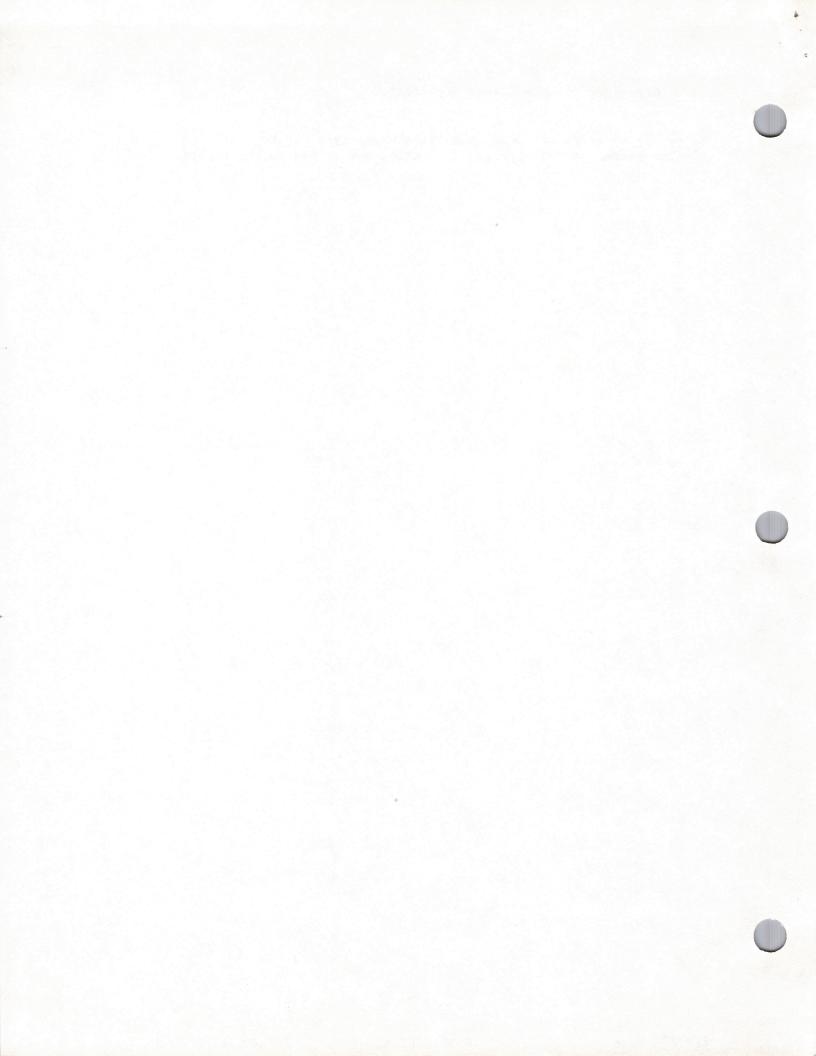
If a file or files are specified for printing, the application should print them in turn, preferably without doing its entire start—up sequence. For example, it may not be necessary to show the menu bar or a document window, and reading the desk scrap into memory is definitely not required. After successfully printing a document, the application should set the file type in the Finder information to \emptyset . Upon return from the application, the Finder will start up other applications as



10 Structure of a Macintosh Application

necessary to print any remaining files whose type was not set to \emptyset .

*** The Finder doesn't currently do this, but it may in the future.



GLOSSARY

bundle: A resource that maps local IDs of resources to their actual resource IDs; used to provide mappings for file references and icon lists needed by the Finder.

Desktop file: A resource file in which the Finder stores folder resources and the version data, bundle, icons, and file references for each application on the volume.

file reference: A resource that provides the Finder with file and icon information about an application.

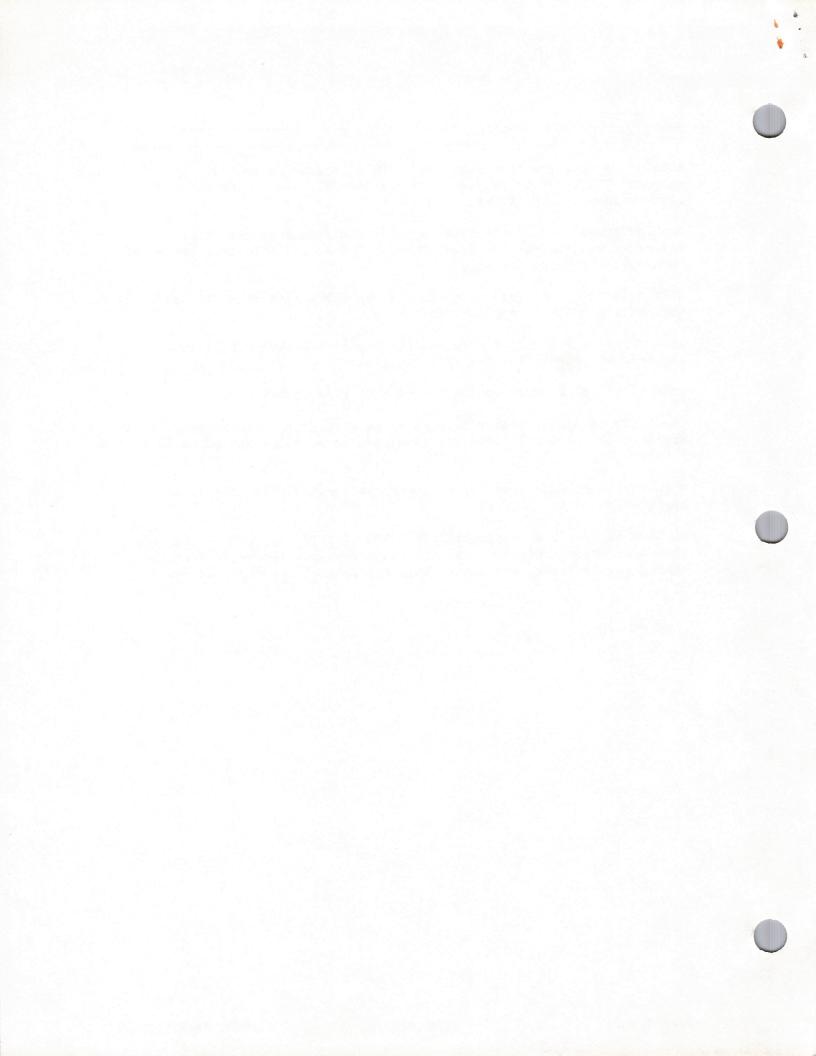
file type: A four-character sequence, specified when a file is created, the identifies the type of file.

icon list: A resource consisting of a list of icons.

local ID: A number that refers to an icon list or file reference in an application's resource file and is mapped to an actual resource ID by a bundle.

signature: A four-character sequence that uniquely identifies an application to the Finder.

version data: In an application's resource file, a resource that has the application's signature as its resource type; typically a string that gives the name, version number, and date of the application.



COMMENTS?

Macintosh User Education encourages your comments on this manual.

- What do you like or dislike about it?
- Were you able to find the information you needed?
- Was it complete and accurate?
- Do you have any suggestions for improvement?

Please send your comments to the author (indicated on the cover page) at 10460 Bandley Drive M/S 3-G, Cupertino CA 95014. Mark up a copy of the manual or note your remarks separately. (We'll return your marked-up copy if you like.)

Thanks for your help!



MACPAINT FORMAT

The MacPaint Document Format by Bill Atkinson

MacPaint documents are easy to read and write, and have become a standard interchange format for full-page bitmap images on MacIntosh. Their internal format is described here to aid program developers in generating and reading MacPaint documents.

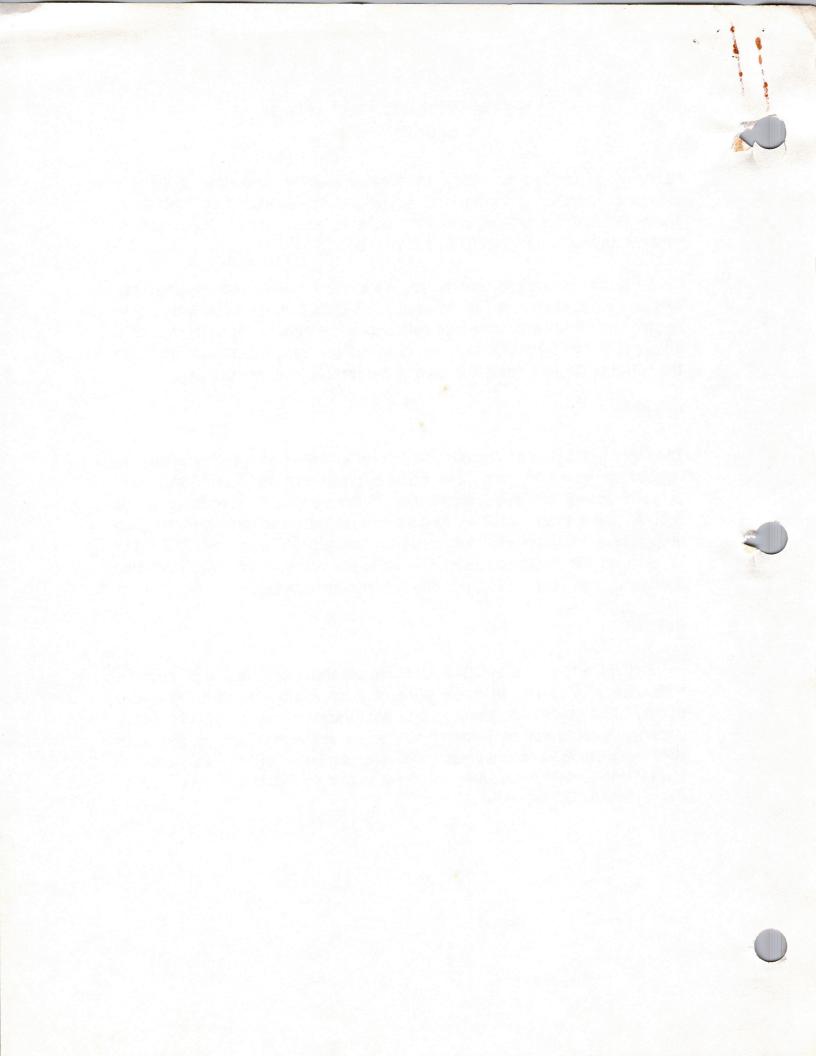
MacPaint documents use only the data fork of the file system; the resource fork is not used and may be ignored. The data fork contains a 512 byte header and then the compressed data representing a single bitmap of 576 pixels wide by 720 pixels tall. At 72 pixels per inch, this bitmap occupies the full 8 by 10 inch printable area of the Imagewriter printer page.

HEADER:

The first 512 bytes of the document form a header with a 4 byte version number (default = 2), then 38*8 = 304 bytes of patterns, then 204 unused bytes reserved for future expansion. If the version number is zero, the rest of the header block is ignored and default patterns are used, so programs generating MacPaint documents can simply write out 512 bytes of zero as the document header. Most programs which read MacPaint documents can simply skip over the header when reading.

BITMAP:

Following the header are 720 compressed scanlines of data which form the 576 wide by 720 tall bitmap. Without compression, this bitmap would occupy 51840 bytes and chew up disk space pretty fast; typical MacPaint documents compress to about 10 Kbytes using the PackBits procedure in the MacIntosh ROM to compress runs of equal bytes within each scanline. The bitmap part of a MacPaint document is simply 720 times the output of PackBits with 72 bytes input.



READING SAMPLE:

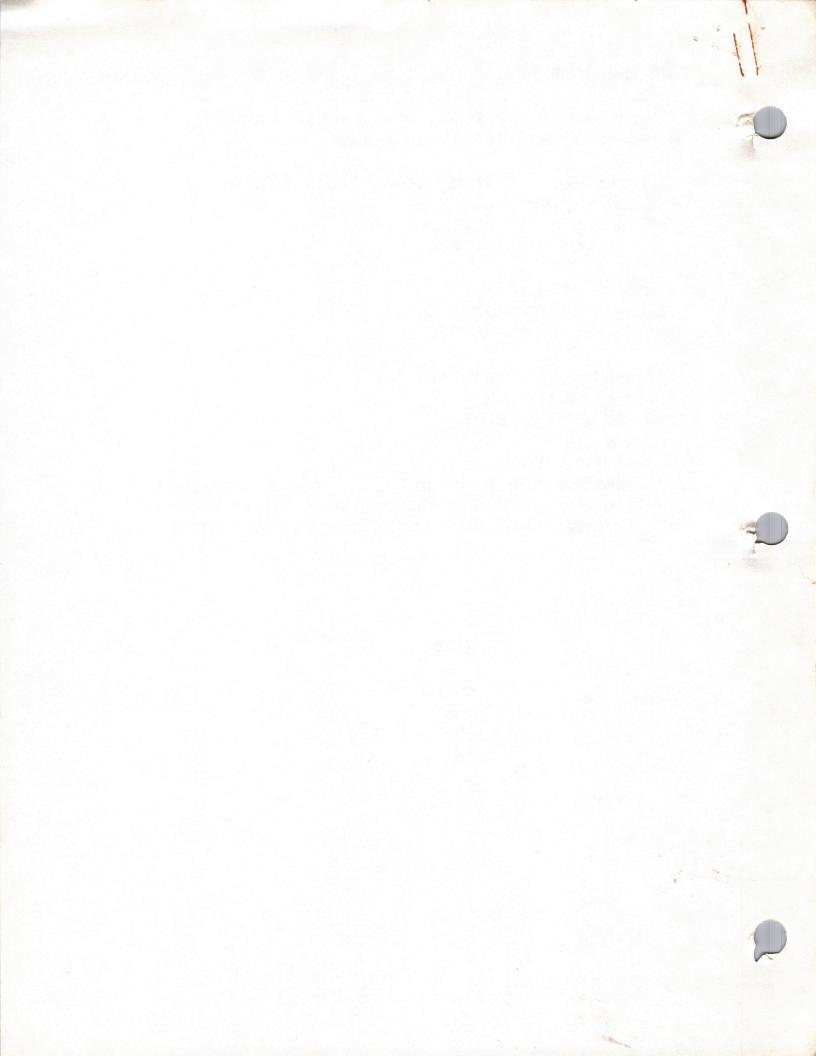
```
CONST srcBlocks = 2; (at least 2, bigger makes it faster)
      srcSize = 1024; [512 * srcBlocks]
TYPE diskBlock = PACKED ARRAY[1..512] OF QDByte:
VAR srcBuf: ARRAY[1..srcBlocks] OF diskBlock;
      srcPtr,dstPtr. QDPtr:
(skip the header)
ReadData(srcFile,@srcBuf,512);
(prime srcBuf)
ReadData(srcFile,@srcBuf,srcSize);
(unpack each scanline into dstBits, reading more source as needed)
srcPtr := @srcBuf;
dstPtr := dstBits.baseAddr;
FOR scanLine := 1 to 720 DO
   BEGIN
      UnPackBits(srcPtr,dstPtr,72);
                                               (bumps both ptrs)
      [ time to read next chunk of packed source ? ]
      IF ORD(srcPtr) > ORD(@srcBuf) + srcSize - 512 THEN
         BEGIN
            srcBuf[1] := srcBuf[srcBlocks];
                                               (move up last block)
            ReadData(srcFile,@srcBuf[2],srcSize-512);
            srcPtr := Pointer(ORD(srcPtr) - srcSize + 512);
         END;
  END;
```



WRITING SAMPLE:

To write out a 576 by 720 bitmap which is contained in memory, the following fragment of code could be used:

```
PACKED ARRAY[1..512] OF QDByte;
TYPE diskBlock =
VAR srcPtr,dstPtr.
                       QDPtr;
      dstBuf:
                       diskBlock;
      dstBytes:
                       INTEGER;
( write the header, all zeros )
FOR i := 1 to 512 DO dstBuf[i] := 0;
WriteData(dstFile,@dstBuf,512);
[ Compress each scanline and write it ]
srcPtr := srcBits.baseAddr;
FOR scanLine := 1 to 720 DO
   BEGIN
      dstPtr:=@dstBuf;
      PackBits(srcPtr,dstPtr,72);
                                                (bumps both ptrs)
      dstBytes := ORD(dstPtr) - ORD(@dstBuf);
                                                ( calc packed size )
      WriteData(dstFile,@dstBuf,dstBytes);
                                                ( write packed data )
   END;
```



PASCAL SER. DRIVER

To:

Macintosh Software Developers

From:

Martin P. Haeberli

Date:

February 19, 1984

Subject: How to use the Pascal Support Package for the Macintosh

Serial Driver

This note is for experts only. I am assuming that you have read and understood the "Async Driver" section of the Inside Macintosh documentation. We are providing this package to you as a gift, without support. This means no questions, please! I know you will ask them anyway, but please please please go easy on us. This package is absolutely not formally released, supported, or documented. If you have decided to proceed anyway, I encourage you to read at least all of the xxxDefs Pascal source code, and skim the xxxImpl Pascal source code, so that you have a general understanding of how to use this stuff

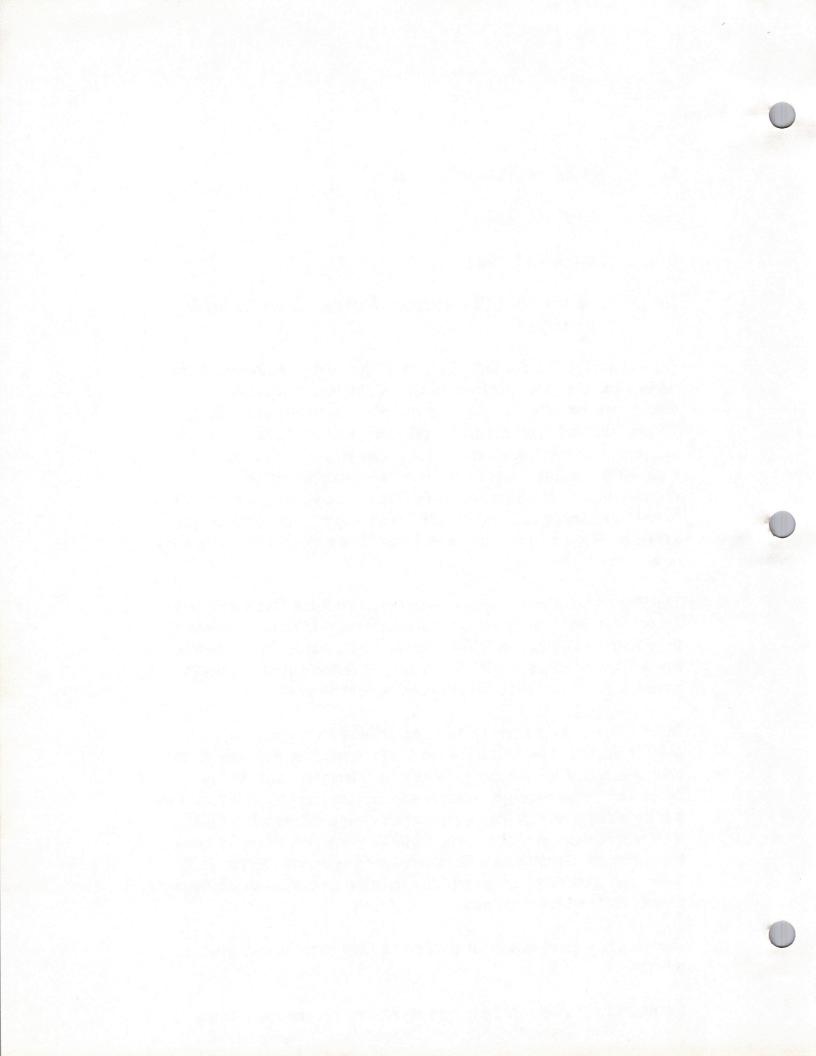
The Macintosh ROM contains a Serial Driver for both Macintosh Serial Ports. This driver supports asynchronous serial I/O only, at speeds from about 150 baud up to speeds over 19.2 K baud. There are also a few known bugs in the ROM-based version of the driver, especially where support for XON-XOFF flow control is concerned.

In developing the Macintosh Terminal Emulator program, MacTerminal, I chose to build a set of Pascal interfaces for the serial driver on top of the Macintosh IO system interfaces published in OSIntf.text. This package, which is implemented mostly in Pascal with a little bit of assembly language, provides the programmer with a RAM-based version of the Async Serial Driver which solves all known bugs, as well as providing support for speeds down to as slow as 50 baud, and providing the opportunity for user extensions to the Async Serial Driver where necessary.

The package comes in a number of parts, here listed in compilation order:

Async.text:

Assembly Language source text for the RAM-based Async Serial Driver. Depends on: SysEqu.text



Execute.text: Assembly Language support for SerImpl and others.

Depends on:

SysEqu.text SysMacs.text

TaskAsm.text: Assembly Language support for TaskImpl. Depends on:

SysEqu.text SysMacs.text

TaskDefs.text: Defines interface to TaskImpl. Depends on:

OSIntf

TaskImpl.text:Pascal Support for simple multitasking system used by Pascal interface to serial driver. Depends on:

QuickDraw OSIntf TaskDefs

PipeDefs.text: Defines interface to PipeImpl. Depends on:

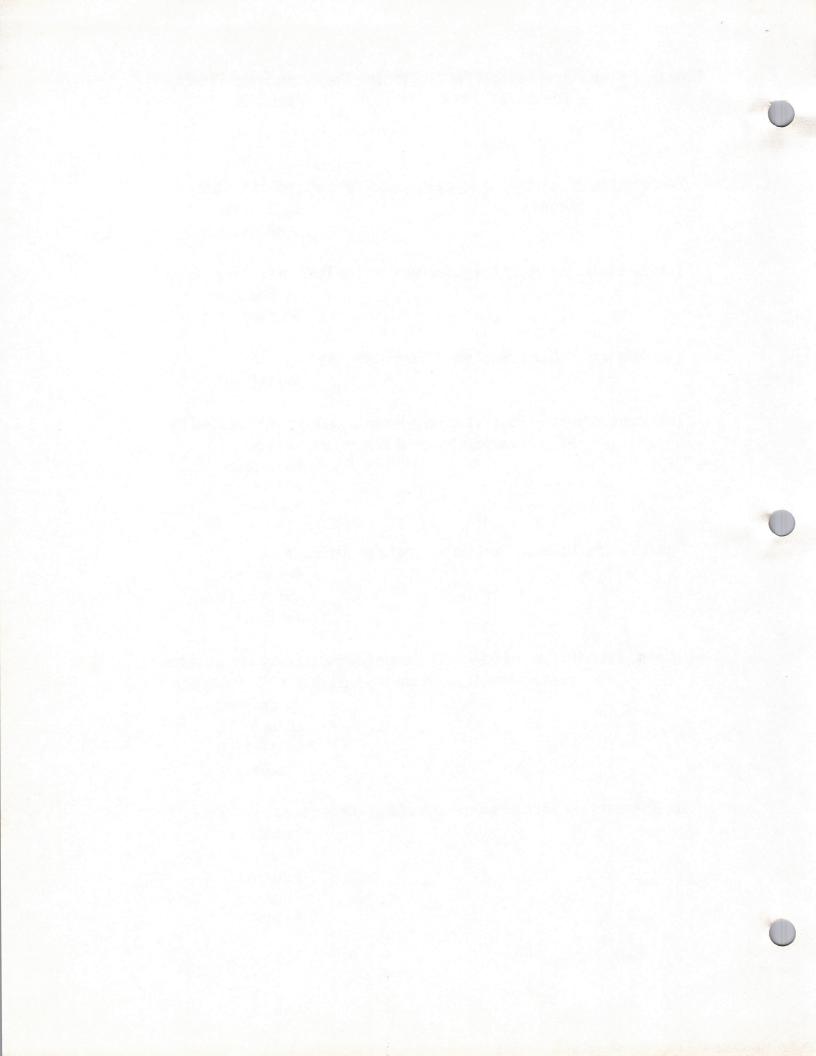
QuickDraw OSIntf ToolIntf

PipeImpl.text: Pascal support for a simple fifo buffer/stream interface used by Pascal Interface to serial driver. Depends on:

QuickDraw OSIntf ToolIntf PipeDefs

SerDefs.text: Defines interface to SerImpl. Depends on:

QuickDraw OSIntf ToolIntf PipeDefs TaskDefs



How to use the Pascal Support Package for the Macintosh Serial Driver.

January 28, 1984 Page 3

SerImpl.text The actual Pascal support for Asynchronous Serial Driver. Knows how to load and install the RAM-based driver, how to unload it, how to set serial line parameters individually, how to take stuff from a pipe and send it to the serial port, and how to take stuff from the serial port and save it in a pipe. Depends on:

QuickDraw OSIntf ToolIntf PipeDefs TaskDefs

To link this in to your program, include the following stub in your linker command file:

obj:OSTraps

obj:ToolTraps

Execute

TaskAsm

Tasklapl

Pipelmpl

. . .

Serlmpl

obj:MacPasLib

Also, you must include the following stub in your rmaker command file in order to include the RAM-based Async driver in your resource file:

Type SERD = WDEF
ASync, 256

Now, for an example of how to use this stuff with your program.

```
Program SerDemo;
       ($U obj:QuickDraw
                                 QuickDraw,
       ($U obj:0SIntf
                                 OSIntf.
       ($U obj:ToolIntf
                                 Tool Intf.
       $U PipeDefs
                                 PipeDefs,
       $U TaskDefs
                                 TaskDefs,
       $U SerDefs
                           }
                                 SerDefs;
Const
   SInPipeSize = 512;
   SOutPipeSize = 256;
Var
   done:
             Boolean;
   pipeSin: Pipe;
                           { Serial input pipe: characters arrive here }
   pipeSOut: Pipe;
                           { Serial output pipe: send characters here }
   serPort: Ser;
                           { Serial Object pointer }
             Char;
   ch:
                           { temporary character }
   theEvent: EventRecord; { used by GetHextEvent below }
   Procedure InitSerDemo:
   Begin
      InitTasks;
                          { Initialize multitasking package }
       { open an input pipe for SInPipeSize characters }
      pipeSIn := PipeOpen(SInPipeSize);
      { open an output pipe for SOutPipeSize characters }
      pipeSOut := PipeOpen(SOutPipeSize);
      InitSer;
                          { Initializes the serial package }
      serPort := NewSer; { Allocates a serial object }
      { Open serPort onto Serial Port A (telephone port) }
      OpenSer(serPort, SPortA, pipeSIn, pipeSOut);
      { choices for settings stuff are listed in SerDefs }
      SetSerSpeed(serPort, SSp9600); { set speed to 9600 }
      SetSerParity(serPort, SParOdd); { set parity to Odd }
      { set character width to 8 bits/character }
      SetSerWidth(serPort, SWid8);
      SerSerStop(serPort, SStp1);
                                   { set no. of stop bits to 1 bit }
  End;
  Procedure EndSerDeno;
  Begin
      CloseSer(serPort);
                                { close the serial port }
      serPort := Nil:
      PipeClose(pipeSIn);
                                 { close the input pipe }
      PipeClose(pipeSOut);
                                 { close the output pipe }
     EndSer;
                                 { Shut down the serial package }
      EndTasks;
                                 { Shut down aultitasking }
  End;
```

400

and deciminate Series

How to use the Pascal Support Package for the Macintosh Serial Driver.

January 28, 1984 Page 5

```
Begin
    InitSerDeno;
   done := False;
   While Not Done Do
       Begin
              SystemTask;
                           { This runs the simple multitasking system }
              RunTasks;
              While PipeAvail(pipeSIn) > 0 Do
                    Begin
                           ch := PipeGet(pipeSIn); { ch is next char }
                            { process char here or elsewhere }
                    End;
             While PipeLeft(pipeSOut) > 0 Do
                    Begin
                           { generate next char }
                           PipePut(pipeSOut, ch); { ch is char to send }
                    End;
              If GetNextEvent(everyEvent, theEvent) Then
                    Begin
                           { handle events as you like here. }
                    End;
      End:
   EndSerDeno;
End.
```

